Web Content Delivery Optimization

Hossein Bayatpour

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Communications Engineering.
Espoo 1.08.2016

Thesis supervisors:

Prof. Jukka Manner

Thesis advisor:

M.S.c Gautam Raj Moktan

Aalto University
School of Electrical Engineering

Author: Hossein Bayatpour

Title: Web Content Delivery Optimization

Date: 01.08.2016 Language: English Number of pages: 6 + 55

Department of Department of Communications and Networking (Comnet)

Professorship: Networking Technology

Supervisor: Prof. Jukka Manner

Advisor: Gautam Raj Moktan

Milliseconds matters, when they're counted. If we consider the life of the universe into one single year, then on 31 December at 11:59:59.5 PM, "speed" was transportation's concern, and now after 500 milliseconds it is web's, and no one knows whose concern it would be in coming milliseconds, but at this very moment; this thesis proposes an optimization method, mainly for content delivery on slow connections. The method utilizes a proxy as a middle box to fetch the content; requested by a client, from a single or multiple web servers, and bundles all of the fetched image content types that fits into the bundling policy; inside a JavaScript file in Base64 format. This optimization method reduces the number of HTTP requests between the client and multiple web servers as a result of its proposed bundling solution, and at the same time optimizes the HTTP compression efficiency as a result of its proposed method of aggregative textual content compression. Page loading time results of the test web pages; which were specially designed and developed to capture the optimum benefits of the proposed method; proved up to 81% faster page loading time for all connection types. However, other tests in non-optimal situations such as webpages which use "Lazy Loading" techniques, showed just 35% to 50% benefits, that is only achievable on 2G and 3G connections (0.2 Mbps – 15 Mbps downlink) and not faster connections.

Keywords: HTTP Request Minimization, HTTP Compression, Base64, Bundling, IUC

Preface

I want to thank Professor Jukka Manner for all I have learnt from him and all I have learnt by his help; which has been drawn briefly inside pages of this thesis. I would also like to thank the brilliant Gautam Raj Moktan and Nutti Varris who helped me to deepen my knowledge around networking and web technologies. Finally, my endless thank goes to my mom “Zhila”, my father “Kavoos”, and my wife, “Setareh”; for their unconditional love and support.

Otaniemi, 01.08.2016

Hosi Bayatpour

Contents

Abstract ii

Preface iii

Contents iv

Symbols and Abbreviations vi

1 Introduction 1

2 Background 3

2.1 Quality of Service 3

2.1.1 Mobile Quality of Service 4

2.1.2 Mobile QoS Limitations 4

2.2 Standard Solutions 5

2.2.1 HTTP Compression 5

2.2.2 Minimizing Number of HTTP Requests 12

2.2.3 Server-Side Caching 16

2.2.4 Content Delivery Networks 20

2.3 Commercial Solutions 21

2.3.1 Akamai 22

2.3.2 CloudFlare 23

2.4 Vendor-specific Solutions 24

2.4.1 Google Data Saver 24

2.4.2 Opera Mini 25

2.5 Academic Ideas 26

2.5.1 Jukka Manner and Le Wang “Web in a Bundle” 26

2.5.2 MIT JavaScript-based Polaris 27

3 Research Methodology, and Experimental Setup 29
3.1 Base64 29
3.2 Aggregative HTTP Compression of Images in a JS Bundle 31
3.3 Methodology Description 31
3.4 Experimental Setup 33
4 Measurement Results, and Analysis 37
4.1 Wi-Fi Connection Test and Analysis 37
4.2 Ethernet Connection Test and Analysis 39
4.3 Cross Platform Test and Analysis 41
4.4 Non-Optimal Case Test and Analysis on 2G, 3G, 4G 43
4.5 Data Transfer Efficiency Analysis 45
4.6 SEO Affection Analysis and Considerations 46
5 Summary 49
References 51

Symbols and Abbreviations

Symbols

KB	Kilo Byte
MB	Mega Byte
ms	millisecond

Abbreviations

2G	2nd Generation of Mobile Telecommunications Technology
4G	4th Generation of Mobile Telecommunications Technology
3G	3rd Generation of Mobile Telecommunications Technology
3GPP	3rd Generation Partnership Project
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CDN	Content Delivery Network
dl	Downlink
CRC	Cyclic Redundancy Check
CSS	Cascading Style Sheet
EEP	Energy Efficient Proxy
EPC	Evolved Packet Core
GIF	Graphics Interchange Format
GGSN	Gateway GPRS Support
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
IUC	Image Unifier and Compressor
JSON	JavaScript Object Notation
LTE	Long-Term Evolution
OECD	Organization for Economic Co-operation and Development
PC	Personal Computer
PCC	Policy and Charging Control
PNG	Portable Network Graphics
QoE	Quality of Experience
QoS	Quality of Service
RCFC	Internet Engineering Task Force's Request for Comments
RNC	Radio Network Control
RTT	Round Trip delay Time
SGSN	Serving GPRS Support
SSL	Secure Sockets Layer
TCP	Transmission control Protocol
TLS	Transport Layer Security
UMTS	Universal Mobile Telecommunications System
URL	Universal Resource Locator
UTRAN	UMTS Terrestrial Radio Access Network
WLAN	Wireless Local Area Network
W3C	World Wide Web Consortium

1 Introduction

From the time “Computer Networks” concept, has been emerged; “Efficient and Optimized Resource Management”, has been always one of the hottest discussion schemes in networking technology communities. Many elements might be involved in a computer network and in a sense, all of them might be recognized as network resources, and they might all fall into one of these well-known resource categories of; “data”, “information”, and “hardware devices”. [1]

As a result, an optimally managed network in terms of resources, might require each and every one of those resources; to be optimally allocated, in order to serve the running services on a network with the highest possible quality. This is where the concept of Quality of Service (QoS) rises. Quality of Service (QoS) refers to the performance of a network bandwidth, latency, error rates, and uptime/downtime probability. [2]

During previous years, the ever increasing amount of mobile devices and users’ interaction with web via their mobile devices, simply demonstrated how considerable the QoS concept might be within the Mobile Networks. Subsequently if a web optimization solution wanted to be remarkably noticed, it should widely consider Mobile Networks and the related limitations such as; “limited capacity”, and “lower throughputs and higher delays than wired networks”. Because of such limitations, “Mobile Networks” will bring more challenges to QoS than other kinds of networks. However, here is not where the story of challenges ends, and that might be where the first words of a long drama have been thrown out.

Since the end users satisfaction has become the key evaluation factor of network services, another concept has appeared beside QoS as the second leg of a deliverable service with the acronym of QoE; which stands for Quality of Experience. QoE can be regarded as a user-focused performance measurement for a given product or service. [3] QoE is all about the experience that the end users are receiving from a delivered service and their satisfaction rate about that. Quality of Experience is becoming more and more vital; as the commerce is moving its place to the web.

Ensuring that the end user on the other end of the connection has a positive experience is vital to maintain or even increase the revenue. From the network perspective, one of the key factors of user experience is the “time to receive” the desired content. Let’s put the term “time to receive” in a more user experience-based format, and call it “time to perceive”. Anything that takes place quicker than 100 milliseconds is perceived as instant by the human perception system; and on the other hand, users can clearly perceive delays that occur between 100 and 300 milliseconds [4] So, even making a web page to load 300 ms faster, might improve the QoE and the QoS. Based on the recent studies on web speed optimization, it has been achieved that 47 percent of web users expect a website to load under two seconds. [5] On the other hand, lack of speed also kills customer conversion rate on mobile, as 74% of users will abandon after waiting five seconds for a mobile site to load, and just to put an end to this paragraph let’s state that; slow loading websites cost retailers £1.73bn in lost sales each year. [6] [7]

Now it might be more obvious that why shorter loading time and smaller transfer size specially on Mobile devices; improves user experience, conversion rate, and search engine rank, beside other benefits such as energy efficiency. In addition, it should be considered that faster web access will also result in creating new services and innovations, and subsequently will lead to more employment and economic growth. Each time a country doubles its internet broadband speed, economic output increases by 0.3%. That may not sound like much, but for the club of rich countries known as the OECD that can be equal to $126 billion every year, or more than 14% of the average annual growth rate of those countries during the previous decade of human history. [8]

But the path might not always be easy and smooth, because improving broadband requires huge infrastructure investments to be built. But it is also possible to bring the faster internet without spending huge amounts of money on infrastructures and building out wires or wireless stations. Application-based solutions will cost less and will be implemented faster. Developing countries, are less dependent on the digital economy, and subsequently are unlikely to see the large jumps in productivity that are driving growth elsewhere. Lack of infrastructural investments has made them to miss lots of economic opportunities. Cheaper solutions may give them more opportunities to grow their economy, earn, and spend for their networks broadband infrastructures. Previous and current infrastructural and application-based attempts to speed up and optimize the web experience has led to numerous of solutions such as; “Content Delivery Networks (CDNs)”, “Server Side Caching”, “HTTP Compression schemes”, and “HTTP Requests Minimization method”.

This thesis intends to propose a new “HTTP Request Minimization method” to optimally manage the delivery of the “Image Type Web Content” as a network resource, to improve the network QoS and QoE; while making the “HTTP Compression schemes” more efficient in terms of image data type compression. The method uses Base64 encoding to convert webpage images into textual data and bundles them in a JavaScript file in order to optimize the image content type delivery by reducing the number of requests for them, and improving their compression efficiency. The test results have shown page loading time improvement up to 81% on all type of connections and in optimal situations where all images of the webpage fits into the bundling policy and get a high compression ratio. In non-optimal cases where there are not enough number of images to be bundled, then the benefits will be small or even not available at all. In such not optimal cases, the tests results have shown between 35 % to 50 % improvement only on slow and decent connections such as 2G, and 3G. (0.2 Mbps – 15 Mbps downlink)

In this thesis; standard, commercial, and vendor-specific, “web content delivery optimization” solutions will be explained in the background chapter, in order to provide a big picture of the problem and available solutions. Some academic ideas about web content delivery optimization is also introduced as the last section of the background chapter. Afterwards, the proposed methodology by this thesis; which is called IUC (Image Unifier and Compressor), is explained through the technology it applies and the requirements for its experimental setup. Finally, the tests result on optimal and non-optimal webpages is reported and analyzed to draw a fare conclusion for the IUC method.

2 Background

Although web technologies and web infrastructures are improved during the previous years, web content delivery optimization has been remained still as a challenge because the amount of deliverable content on web is growing as well. In this chapter the origin of the need for faster user experience and the proposed solutions for that are discussed in order to draw a background big picture of the web content delivery optimization.

2.1 Quality of Service

When looking for the meaning of quality in dictionaries, two different definitions will be found in most of them. One is “how good or bad something is”, and the other one is “a high standard”. [9] When it comes to “Computer Networks”, the definition of “Quality” is pretty close to “a high standard”, and it’s mostly used in conjunction with the word “Services”; which in computer networks means, the functions provided by the network infrastructure in order to offer value to the business process. [10] Having these two words beside each other creates the term Quality of Service (QoS) which can be regarded as an industry-wide set of standards and mechanisms for ensuring high quality performance for applications to achieve maximum bandwidth and deal with other network performance elements like latency, error rate and uptime. [11]

By using QoS mechanisms, network administrators can use existing resources efficiently and ensure the required level of service without reactively expanding or over provisioning their networks. Traditionally, the concept of quality in networks meant that all network traffic was treated equally. The result of that traditional approach was that all network traffic received the network’s “best effort”, with no guarantees for reliability, delay, variation in delay, or other performance characteristics. With “best effort” delivery service, a single “bandwidth intensive” application can result in poor or unacceptable performance for all other applications sharing the same bandwidth. The QoS concept of quality is the one in which the requirements of some applications and users are more critical than others, which means that some traffic needs preferential treatment. [12]

Network administrators can use QoS to guarantee throughput for applications so that their transactions can be processed in an acceptable amount of time. QoS provides the following benefits:

- Gives administrators control over network resources and allows them to manage the network from a business, rather than a technical, perspective.
- Ensures that time-sensitive and mission-critical applications have the resources they require, while allowing other applications access to the network.
- Improves user experience.
- Reduces costs by using existing resources efficiently, thereby delaying or reducing the need for expansion or upgrades.

2.1.1 Mobile Quality of Service

In wireless mobile networks QoS refers to the measurement of a system with good transmission quality, service availability and minimum delay. Cellular systems by nature have finite resources. Radio spectrum and transport resources are limited, expensive, and shared between many users. Mobile broadband networks must support multiple applications of voice, video, and nowadays mostly data on a single IP-based infrastructure. Each of these services which are converged together in a mobile network; has its own unique traffic handling and QoS requirements. Such issues cannot be economically solved by over provisioning the network, and might need other application-based solutions.

In mobile networks, a positive user experience must be obtained through efficient use of the available wireless network resources. The 3rd Generation Partnership Project (3GPP), the Universal Mobile Telecommunications System (UMTS) and Long-term Evolution (LTE) standards body, has developed a comprehensive QoS, charging, and policy control framework to address this problem. The policy and charging control (PCC) is the heart of the Evolved Packet Core (EPC), and ensures QoS for a particular subscription and service type. In 4G it is expected to have a reliability of at least 99.999 referred to as five nine reliability. [13]

Also granular control of service quality is critical for operators to establish new business models and monetize services. It will enable the operators to employ “fair use” policies that limit subscriber service abuse, and maintains network performance during peak traffic times. Before spending billions of dollars on equipment and deployments, forward thinking operators will carefully evaluate vendors and proactively measure the QoS and policy management functions of their network devices.
But the question is that, “Is limiting users by “fair-use policies” is enough?”. Operators can also optimize the deliverable data by applying “fair-content optimization policies” such as the proposed method in this thesis, beside their available “fair-use policies”.

2.1.2 Mobile QoS Limitations

In 1G networks and 2G networks such as GSM and CDMA there was only one aspect of QoS and it was voice. Providing quality speech was the major concern. Now in 3G and 4G networks QoS has to be provided for voice as well as data. The major challenges when considering QoS in cellular networks are varying rate channel characteristics, bandwidth allocation, fault tolerance levels and handoff support among heterogeneous wireless networks. It is fortunate that each layer which includes physical, MAC, IP, TCP and application have got their own mechanisms to provide QoS. It is important to guarantee QoS in each layer so that the network is more flexible and tolerant to QoS issues.

But beside all the QoS mechanisms in different network layers; it should also be considered that bandwidth allocation plays a major role in Mobile QoS. It must be made sure that bandwidth is allocated in an efficient manner and also the remaining bandwidth should not be wasted. Some schemes like Renegotiation scheme takes care of this issue by allocating the remaining bandwidth to lower priority classes. Things get even more complicated when data and voice service has to be supported. Voice services are very

delay sensitive and require “real time” service. On the other hand, data services are less delay sensitive but are very sensitive to loss of data and also they expect error free packets. So both these factors have to be considered for providing QoS for voice and data services. [15] Optimized allocation of bandwidth for the deliverable content is important, but beside that; delivering the optimized content sounds brilliant. Optimization in transfer size and number of established connections can also help allocation mechanisms to behave more relaxed.

2.2 Standard Solutions

There has been so many attempts for finding solutions towards web content delivery optimization, but only a few of them has been accepted as standard solutions. In this section different standard solutions in form of algorithms, protocols and hardware solutions are explained. Algorithms such as different “HTTP Compression schemes”, protocols such as “SPYDY” and “HTTP2”, and hardware solutions such as “server-side caching”, and “content delivery networks”; are those popularly accepted solutions which are explained within the next sections.

2.2.1 HTTP Compression

Text compression can be briefed into some main steps of; “removing all unneeded characters”, “inserting a single repeat character to indicate a string of repeated characters”, and “substituting a smaller bit string for a frequently occurring bit string”. Compressing data can be a lossless process or a lossy one.

“Lossless” compression enables the restoration of a file to its original state, without the loss of a single bit of data. Lossless compression is the typical approach with executables, as well as text and spreadsheet files, where the loss of words or numbers would change the information. “Lossy” compression permanently eliminates bits of data that are redundant, unimportant or imperceptible. [16] In a lossy compression some part of the whole bunch of data is missed when it’s decompressed, such as some image compression methods which will produce lower quality and lower resolution images, compared to the originals.

The main advantages of compression are the reduction in storage hardware, data transmission time and communication bandwidth, and the resulting cost savings. The main disadvantage of compression is the performance impact resulting from the use of CPU and memory resources to compress and decompress the data. The compression and decompression processes at servers also cost some amount of time which gets especially remarkable as a cost when the user is using a fast connection through which the whole data transfer time gets very small while the time spent on compression and decompression stays the same.

Many vendors have designed their systems in order to try to minimize the impact of the “processor intensive calculations” associated with compression. If the compression runs inline, before the data is written to disk, the system may offload compression to preserve system resources. As an example; IBM uses a separate hardware acceleration card to

handle compression with some of its enterprise storage systems. So, it seems that major market players tend to spend more on hardware optimization in order to achieve more from their software solutions.

Now that the compression concept and its advantages and disadvantages are briefly described, it's the time to talk about HTTP compression. Before that, let's simply explain HTTP (Hypertext Transfer Protocol). Whenever a web browser fetches a file like a webpage or an image, from a web server, it uses a transfer protocol such as HTTP which is a request/response protocol. This means that the client device sends a request for some file and the web server sends back a response message followed by the file itself. If the used transfer protocol is HTTP, then that request which is sent from the computer to the web server; will be an HTTP request and contains all sorts of interesting information such as; "the IP address of the client and/or the client HTTP proxy", "which document the client requested", "which version of which browser the client is using", "which page the client came from to get here", "the client's preferred language", "cookies", and "accepted compression encoding schemes".

Generally speaking, HTTP compression allows the web content to be compressed on the server before transmission to the client. For resources such as "text", this can significantly reduce the size of the response message, which results in reducing the bandwidth requirements and download times. But, some content types such as "png" or "gif" images are already compressed and do not respond well to compression. In fact, attempting to compress them can increase the size of the response message and waste CPU time on the server. Surprisingly, from now on and with the proposed "Aggregative Lossless HTTP Compression" method in this thesis, it has been made possible to collectively compress group of all image content types such as "png" or "gif" up to 75 % in a lossless manner.

Apart from the "Aggregative Lossless HTTP Compression" method; proposed in this thesis, the typical savings on compressed text files may range from 60% to 85%, depending on how redundant the code is. Webmasters who have deployed HTTP compression on their servers report 30 to 50% savings of their bandwidth bills. [17] There are two HTTP compression schemes or algorithms which are commonly used, "GZIP", and "DEFLATE".

HTTP clients indicate their support of compression using the "Accept Encoding" header like: "Accept-Encoding: gzip, deflate". As a result, a server will only compress content for clients that show support for compression. The server will set the "Content Encoding" HTTP header like; "Content-Encoding: gzip", so that the client knows which algorithm to use when reading the response body. Now it's the time to explain the HTTP compression schemes and practices in more details, starting with "GZIP" to raise the interest, and then moving to "DEFLATE" in order to tell where "GZIP" originally comes from.

2.2.1.1 GZIP

GZIP is a generic compression scheme that can be applied to any stream of bytes. Like other compression methods, under the hood it remembers some of the previously seen content and attempts to find and replace duplicated data fragments in an efficient way. However, in practice, GZIP performs best on text-based content, often achieving compression rates of as high as 70 % to 90 % for larger files, whereas running GZIP on assets that are already compressed such as most image of formats yields to no remarkable improvement. The good news is that, from now on, with the proposed "Aggregative Lossless HTTP Compression" method in this thesis, GZIP can gain up to 75% efficiency also in collective image compression.

As a standard scheme, all modern browsers support and automatically negotiate GZIP compression for all HTTP requests. The following table illustrates the savings provided by GZIP compression for a few of the most popular JavaScript libraries and CSS frameworks. The savings range from 60 to 88%, and note that the combination of minified files; which are usually identified by ".min" in their filenames, plus GZIP, MIGHT offer an even larger win in terms of final file size. [18] Table1 shows how different minified and non-minified textual libraries can benefit from GZIP.

Library	Size	Compressed Size	Compression Efficiency
jquery-1.11.0.js	276 KB	82 KB	70 %
jquery-1.11.0.min.js	94 KB	33 KB	65 %
angular-1.2.15.js	729 KB	182 KB	75 %
angular-1.2.15.min.js	101 KB	37 KB	63 %
bootstrap-3.1.1.min.css	118 KB	18 KB	85 %
bootstrap-3.1.1. css	98 KB	17 KB	83 %
foundation-5.css	186 KB	22 KB	88 %
foundation-5.min.css	146 KB	18 KB	88 %

Table1: GZIP Compression efficiency on common libraries

The process of enabling GZIP is one of the simplest and highest optimization methods' implementations and sadly, many people still forget to implement it. Most web servers will compress content on behalf of the admins, and the admins just need to verify that the server is correctly configured to compress all the textual content types that would benefit from GZIP compression.

As an example in order to enable the GZIP on an Apache web server it is only needed to modify the".htaccess" file on the web server with the following scripts and then your server automatically does the compression for any requests that fetch some asset from the folder in which this ".htaccess" file is located. It's also possible to exclude some content types which you don't want to be compressed.

```
<ifModule mod_gzip.c>
mod_gzip_on Yes
mod_gzip_dechunk Yes
mod_gzip_item_include file .(html?|txt|css|js|php|pl)$
mod_gzip_item_include handler ^cgi-script$
mod_gzip_item_include mime ^text/.*
mod_gzip_item_include mime ^application/x-javascript.*
mod_gzip_item_include mime ^image/.*
mod_gzip_item_exclude rspheader ^Content-Encoding:.*gzip.*
</ifModule>
```

"GZIP" is often also used to refer to the GZIP file format, which is:

- A 10-byte header, containing a magic number like “1f 8b”, a version number, and a timestamp
- Some optional extra headers, such as the original file name
- A body, containing a DEFLATE-compressed payload
- An 8-byte footer, containing a CRC-32 checksum and the length of the original uncompressed data, modulo 2^32 [19]

Actually GZIP is based on the DEFLATE algorithm. DEFLATE was intended to be as a replacement of data compression algorithms which at the time, limited the usability of compression.

2.2.1.2 DEFLATE

Before trying to understand DEFLATE in deep, It is vital to understand the other two compression strategies which are used in its construction. Those two are “Huffman coding” and “LZ77 compression”.

A Huffman code is a prefix code prepared by a special algorithm. Just like the prefix numbers in phone numbers. But the difference is that instead of each prefix code being a series of numbers between 0 and 9, in Huffman code each prefix code is a series of bits, either 0 or 1. Instead of each code representing a phone, each code represents an element in a specific “alphabet" such as the set of ASCII characters. A Huffman algorithm starts by assembling the elements of the “alphabet," each one being assigned a “weight" which is a number that represents its relative frequency within the data to be compressed. These weights may be guessed at beforehand, or they may be measured exactly from passes through the data, or some combination of the two. In any case, the elements are selected with a collection of two at a time, and the elements with the lowest weights are being chosen first. [20] The two elements are made to be leaf nodes of a node with two branches. Suppose that there is a set of elements and weights that looked like this:

A 16
B 32
C 32
D 8
E 8

We would pick D and E first, and make them branches of a single node. One of them will be the “0” branch, and the other one will be as the “1” branch.

```
    ( )
0 /   \ 1
 D     E
```

At this point, no element has been given its complete code yet, but at the moment it is clear that the codes for D and E will be exactly the same, except for the last binary digit, in which D will end in 0 and E in 1. Then the combined node of “D/E”, is placed back with the other uncombined elements, and given a weight equal to the sum of its leaf nodes; which in this case is, 8 + 8 = 16.

Again, we take the two nodes with lowest weight, which are A, and “D/E”, and join them into a larger node.

```
      ( )
  0 /   \ 1
  ( )    A
0 /   \ 1
 D     E
```

This time, when the combined node of “A/D/E” is put back into the list of elements, all remaining elements have the same weight, 32; which two of the three are selected to be combined first. When all nodes have been recombined into a single “Huffman tree", then by starting at the root and selecting 0 or 1 at each step, you can reach any element in the tree. Each element now has a Huffman code, which is the sequence of 0's and 1's that represents that entire path through the tree.

Now, it should be fairly easy to see how such a tree, and such a set of codes, could be used for compression. If compressing ordinary text, for example, probably more than half of the ASCII character set could be left out of the tree altogether. Frequently used characters, like “E” and “T” and “A” will probably get much shorter codes, and even if some codes are actually made longer, they will be the ones that are used less often.
However, there is also the question: how do you pass the tree along with the encoded data? It turns out that there is a fairly simple way, if you modify slightly the algorithm used to generate the tree. [20]

In the classic Huffman algorithm, a single set of elements and weights could generate multiple trees. In the variation used by the Deflate standard, there are two additional rules: elements that have shorter codes are placed to the left of those with longer codes. Just note that in the previous example, D and E wind up with the longest codes, and so they would be all the way to the right.

Among elements with codes of the same length, those that come first in the element set are placed to the left. (If D and E end up being the only elements with codes of that length, then D will get the 0 branch and E the 1 branch, as D comes before E.) It turns out that

when these two restrictions are placed upon the trees, there is at most one possible tree for every set of elements and their respective code lengths. The code lengths are all that we need to reconstruct the tree, and therefore all that we need to transmit.

LZ77 compression works by finding sequences of data that are repeated. The term “sliding window" which is frequently used in this compression method; means that at any given point in the data, there is a record of what characters went before. A 32K sliding window means that both compressor and decompressor have a record of what the last 32768 (32 * 1024) characters were.

When the next sequence of characters to be compressed is identical to one that can be found within the sliding window, the sequence of characters is replaced by two numbers: a distance, representing how far back into the window the sequence starts, and a length, representing the number of characters for which the sequence is identical. [21]

Let us look at some highly compressible data:

Hosi hosi hosi hosi hosi!

The datastream starts by receiving the following characters: “H”, “o” “s”, “i,” “ ”, and “h”. The next five characters will be indexed like:

```
vvvvv
Hosi hosi hosi hosi hosi!
    ^^^^^
```

There is an exact match for those five characters in the characters that have already gone into the datastream, and it starts exactly five characters behind the point where we are now. This being the case, we can output special characters to the stream that represent a number for length, and a number for distance. [23] The data so far is:

Hosi hosi h

The compressed form of the data so far is:

Hosi h[D=5,L=5]

The compression can still be increased, though to take full advantage of it requires a bit of cleverness on the part of the compressor. Look at the two strings that are decided to be identical. Compare the character that follows each of them. In both cases, it's “o”, so we can make the length 6, and not just five. But if we continue checking, we find the next characters, and the next characters, and the next characters, are still identical, even if the so-called “previous" string is overlapping the string that is tried to be represented in the compressed data.

It comes to the point that those 18 characters which start at the second character place, are identical to the 18 characters that start at the seventh character place. It's true that when decompressing is being done, and the algorithm identifies the “length” and “distance” pair; it does not know what all those 18 characters will be yet. But if the algorithm put in place the ones that it knows, then it will know more, which will allow to put down more, or, knowing that any “length” and “distance” pair; where length is bigger than distance is going to be repeating characters again and again, it is possible to set up the decompressor to do just that. So the highly compressible data can be compressed down to just this:

Hosi h[D=5, L=18]!

The DEFLATE compressor is given a great deal of flexibility in how to compress the data. The programmer must deal with the problem of designing smart algorithms to make the right choices, but the compressor itself, already have choices about how to compress data for an efficient compression ratio.

There are three modes of compression that the compressor has available:

- Not compressed at all. This is an intelligent choice for, say, data that's already been compressed. Data stored in this mode will expand slightly, but not by as much as it would if it were already compressed and one of the other compression methods was tried upon it.
- Compression, first with LZ77 and then with Huffman coding. The trees that are used to compress in this mode are defined by the Deflate specification itself, and so no extra space needs to be taken to store those trees.
- Compression, first with LZ77 and then with Huffman coding with trees that the compressor creates and stores along with the data. [22]

But from a web server admin perspective nowadays it’s just modifying the “.htaccess” file to include the configuration scripts for DEFLATE. As an instance, the following script, configures an Apache web server to have the DEFLATE compression enabled.

```
<ifModule mod_deflate.c>
AddOutputFilterByType DEFLATE text/plain
AddOutputFilterByType DEFLATE text/html
AddOutputFilterByType DEFLATE text/xml
AddOutputFilterByType DEFLATE application/xml
AddOutputFilterByType DEFLATE application/javascript
AddOutputFilterByType DEFLATE application/x-javascript
</ifModule>
```

2.2.2 Minimizing Number of HTTP Requests

Making a single HTTP requests and receiving a response for that will cost some time. Imagine there is a webpage which contains 150 images and 3 CSS files and 2 JS files, then the client needs to make 156 different successful HTTP requests and 150 subsequent responses to initiate downloading all the page elements and hope to load them successfully.

So eliminating some of the HTTP requests will eliminate their RTT "Round Trip delay Time"; which from the signaling perspective is the length of time it takes for a signal to be sent plus the length of time it takes for an acknowledgment of that signal to be received. This RTT time for each request, is a part of the total page load time. So reducing and minimizing the number of HTTP requests will directly result in reducing the total page loading time.

There are a lot of common practice for minimizing number of HTTP requests such as:

- CSS Files Unification: Combining all CSS data and files into one CSS file.
- JS Files Unification: Combining all JavaScript files into one JS file.
- Inline Styling: Using styles in HTML page instead of an external CSS file.
- Inline Images: Using the data URL scheme to embed the image data.

In the next sections more advanced methods and protocol will be discussed to clarify the scale of the research area.

2.2.2.1 SPYDY

With regard to the definition from its owner Google, "SPDY is a networking protocol whose goal is to speed up the web. SPDY augments HTTP with several speed-related features that can dramatically reduce page load time". [23] The general solutions offered by SPDY can be shortlisted as:

- SPDY allows multiple, simultaneously multiplexed requests over a single connection, saving on round trips between client and server, and preventing low-priority resources from blocking higher-priority requests.
- SPDY allows the server to actively push resources to the client that it knows the client will need (e.g. JavaScript and CSS files) without waiting for the client to request them, allowing the server to make efficient use of unutilized bandwidth.
- SPDY allows client and server to compress request and response headers, which cuts down on bandwidth usage when the similar headers (e.g. cookies) are sent over and over for multiple requests.

Standard HTTP needs to make a new TCP request for all the objects on the page, because there is significant overhead for each new TCP connection, all these connections slow down performance significantly. SPDY's biggest win comes from "multiplexing". This means that multiple objects from a particular website are requested and retrieved from a single request. Less connection overhead means faster page loads.

HTTP also requests objects in a particular order and one slow resource can block the loading of other resources. SPDY allows the browser to query for multiple objects in one request and for the objects to be sent down the wire as they are ready and out of order. Again, this can increase performance by not holding up the delivery of objects that are available quickly because some take longer to request.

SPDY includes some other performance wins as well. It allows HTTP headers to be compressed, which isn't possible with standard HTTP connections. The compression algorithm uses an HTTP equipped dictionary, which means common strings that appear in headers don't need to be sent across the network. Every byte that is not needed to be sent, not only reduces bandwidth use but also, increases web performance.

While there is a lot of excitement around SPDY, there are some limitations. The first is that only certain browsers support the protocol. While the Internet Engineering Task Force (IETF) is considering SPDY as an official Internet protocol, it has not yet been adopted so it's not clear whether there will be additional support from browsers such as Microsoft's Internet Explorer and Apple's Safari. SPDY is built on top of TLS, which means it requires a site to have a valid SSL certificate in order to work. This, unfortunately, limits SPDY only to CloudFlare's paid customers who have enabled Flexible or Full SSL support. Microsoft is working on revised IETF proposal that is SPDY-like, but removes the requirement for SSL/TLS. If the TLS requirement is removed in the future, it will be available more broadly even for non-TLS connections.

Very few sites currently support SPDY. Google's sites and Twitter being the most notable to support the protocol. As a result, there haven't been a significant number of real world case studies. A recent article by an Akamai researcher pointed out that for much of the web SPDY's performance wins will be limited. [24] The primary reason for this caveat is that most sites are a collection of objects from multiple sources. Since SPDY's multiplexing only works on a per-domain basis, if a site has objects pulled from multiple sources then even if all those sources support SPDY connections still won't be able to be multiplexed between them.

Some CDN providers like the mentioned “CloudFlare” company, has used SPDY on their proxy servers and called it “Rocket Loader”. The CDN provider proxy servers will act as a gateway. An origin server doesn't need to support SPDY. Instead, visitors with browsers that support SPDY connect to proxy over the protocol. The proxy handles the multiplexing and begin sending down objects that it already has in its cache. The request to the origin server for non-cached objects is sent over standard HTTP/S. “CloudFlare” has claimed that the “Rocket Loader” is built to provide multiplex-like support over a standard HTTP connection. By gathering all scripts, regardless of where they're hosted, into a single HTTP request, Rocket Loader limits the number of HTTP connections that are needed. As final words, SPDY has perceived complicated from web owners point of view to be set up, and more than that support on most web servers is unproven. Because of this, websites and web owners have been hesitant to setup SPDY support themselves. Minor changes on SPDY secure connection requirements might result in more popularity of it in future.

2.2.2.2 HTTP2

In previous section, it was explained how SPDY enjoyed some success, gaining adoption with both servers and browsers. However, despite Internet Explorer 11 being supported, Microsoft's Edge browser has dropped it, due to Microsoft implementing support for HTTP/2, the latest version of the HTTP protocol. While other current browsers still maintain support for SPDY, Chrome will remove support in 2016, and other browsers will likely follow.

At the time of writing this thesis, Edge, Firefox, Chrome and Opera support both SPDY and HTTP/2. Safari, including on iOS, will join that group later this year with the launch of Safari 9. HTTP/2 builds on the success of SPDY, which was used as a starting point for the new protocol. As such, the majority of SPDY's objectives are met in HTTP/2. The requirement for an HTTPS connection has been dropped. That said, all of the browser vendors have decided to only implement HTTP/2 for TLS (https) connections.

So while HTTP/2 could be used potentially with clear text in server to server communications, the use case of serving HTTP/2 to browsers means that it is needed to have the web site running on https before the client can even think of moving to HTTP/2. The HTTP/2 specification was finalized in February 2015; a year on, browser support in modern browsers is excellent. As with SPDY, HTTP/2 requires support both on the browser and server level, and there are already many web server implementations. Here are some of the improvements offered by HTTP2:

- HTTP/2 is binary, instead of textual.
- It is fully multiplexed, sending multiple requests in parallel over a single TCP connection.
- It uses header compression HPACK "Header Compression for HTTP/2" to reduce overhead.
- It allows servers to "push" responses proactively into client caches instead of waiting for a new request for each resource
- It uses the new ALPN "Application-Layer Protocol Negotiation" extension which allows for faster-encrypted connections since the application protocol is determined during the initial connection.
- It reduces additional round trip times (RTT), making your website load faster without any optimization.
- Domain Sharding and asset concatenation is no longer needed with HTTP/2. [25]

"Inlining Assets" or "Assets Concatenation" is the practice of embedding CSS stylesheets, external JavaScript files, and images directly into an HTML page. "Domain Sharding" is a common technique for tricking browsers into opening more TCP connections than they're supposed to. Browsers limit the number of connections to a single origin server, but by splitting your website's assets across multiple domains, you can get it to open an extra set of TCP connections. It's a big mistake to consider the advent of HTTP2 as an end to all optimization attempts done by web developers. Optimization for HTTP/2 requires a different mindset. Instead of worrying about reducing HTTP requests, web developers should focus on tuning their website's caching behavior.

The general rule is to ship small, granular resources so they can be cached independently and transferred in parallel; which is called “Multiplexing”. Multiplexing lets multiple requests share a single TCP connection, allowing several assets to download in parallel without the unnecessary overhead of establishing multiple connections. This eliminates the head-of-line blocking issue of HTTP/1.1. Header compression further reduces the penalty of multiple HTTP requests, since the overhead for each request is smaller than the uncompressed HTTP/1.1 equivalent. The following figure illustrates how multiplexing is handled in HTTP2.

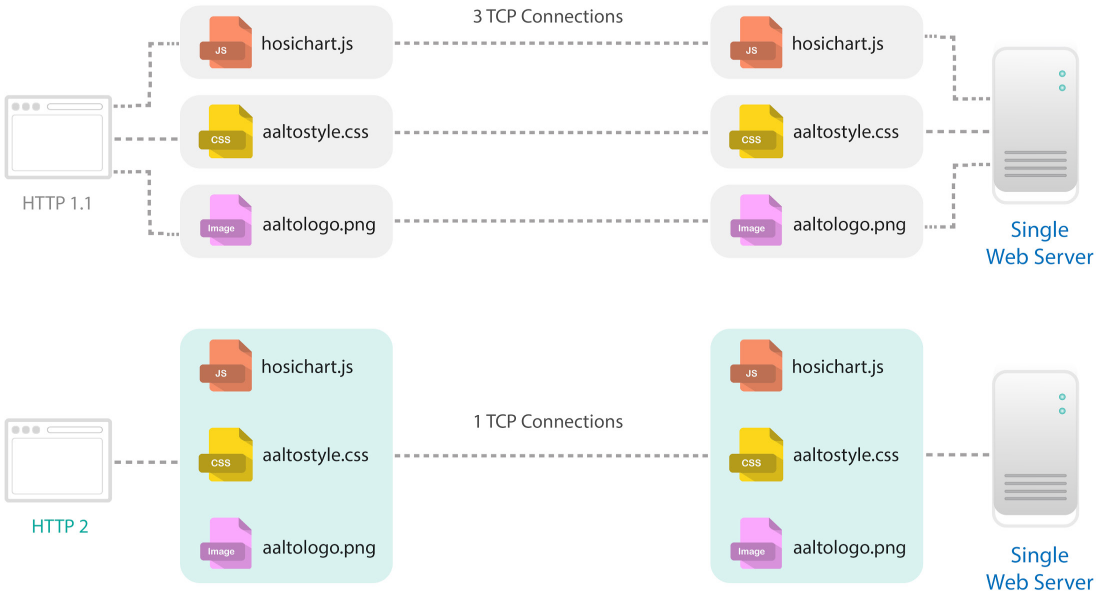

Figure 1: HTTP 2 Multiplexing

Previous optimization methods such as; “DNS Lookups Reduction”, “Content Delivery Network Implementation”, “Browser Caching Leverage”, “HTTP Requests and Responses Size Minimization”, and “Unnecessary Redirects Elimination”, are still required even with HTTP2, but some more optimization considerations are also added; which are considered as limitations by the author of this thesis, such as; “Avoiding Inlining Assets”, and “Avoiding Domain Sharding”.

Beside the TLS connection requirement of HTTP2 as a heritage from SPDY and other optimization limitations, the most important limitation is that; HTTP2 is functioning at its highest efficiency rate when there is a single web server at the other end. Briefly speaking. The multiplexing only happens for the requests and response to and from a single web server, and not multiple web servers. This is the bigger picture of what is previously discussed as “Avoiding Domain Sharding” limitation. It could be possible to use a proxy as a middle box which uses HTTP2, in order to multiplex all requests through one TCP connection for the client.

CloudFlare as web optimization solution provider has implemented HTTP2 on its servers by the time of writing this thesis, and surprisingly HTTP2, seems to be always slower than HTTP1 in loading their DEMO test in Chrome on Windows. The Following screenshot shows one of the tests done with Mozilla Firefox 45.2.0 through a LAN connection on 19.7.2016 on a Windows 7 Aalto University PC with the IP address of 130.233.145.163. While on another test; done with Safari on iMac, CloudFlare HTTP2 implementation gave a remarkable improvement of 80 % on image load time, which makes the CloudFlare HTTP2 demo sounds unreliable.

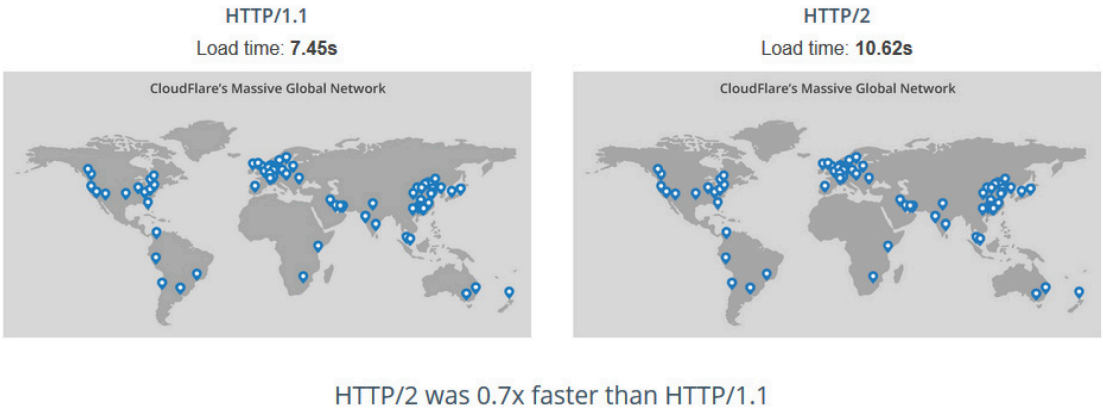

Figure 2: CloudFlare's HTTP2 Demo Test screenshot in Mozilla Firefox on Win7 PC

2.2.3 Server-Side Caching

A cache can be distinguished as a temporary library of files, such as HTML documents, designed and optimized for fast read-and-write access to "short lived" data. Clients can receive frequently requested static HTML files much faster if they have been stored in a specific "web content cache". In the early days of the World Wide Web, the Internet consisted of primarily static information. Users shared fixed information over a public network. Dial-up connections were slow and online traffic was light. [26] Since those early years, the Internet has changed dramatically and continues to change as content becomes increasingly dynamic. Powerful Web servers are now required to handle the increasing demands from users. Additionally, bandwidth capacity continues to improve as users demand instant access to entire business and entertainment applications through their browsers. With these ever increasing needs, "server side caching" emerges as the best response to the performance requirements of the modern heavy weighted websites. Large number of images and graphics on webpages are growing as users find visual content faster to perceive and more attractive to explore.

HTML and graphic files represent static content because they look identical each time users download them into their browsers. Dynamic content is displayed information that can change with every user request. Nowadays' web advertisements and their included images might be a good example of Dynamic content; which might be changed from one user to another based on their personal interests. This content is based on a set of instructions executed on the server side. Information may change each time it is displayed, depending on the code that is being executed.

Caching allows static files to be quickly located in the cache rather than downloaded each time from the Web site. The same approach applies if consecutive requests for a particular dynamic page produce identical results. In this case, output also can be stored in a cache and set aside for future use. Rather than reproducing the HTML version for each request, the server can send the page from its cache, which avoids repetitive work and saves processor cycles for more productive work. Forward and reverse caching are familiar techniques used for many Web sites that must meet high performance and ever increasing traffic demands of users. A third option, is server side caching, which is becoming increasingly popular as a dynamic information caching method. [27]

2.2.3.1 Forward Proxy Caching

As an old standard, forward proxy servers are supposed to be implemented on the edge of a network to improve the delivery of external Web content to internal users of the network. The proxy cache stores and delivers the most frequently accessed content from the millions of documents on the Web. This technique provides better quality of service for end users and on the other side of the game reduces the networking cost necessary to retrieve these documents from the origin server. So, it sounds like a win-win game for users and web owners. [27] Figure 3 explains caching in a Forward proxy.

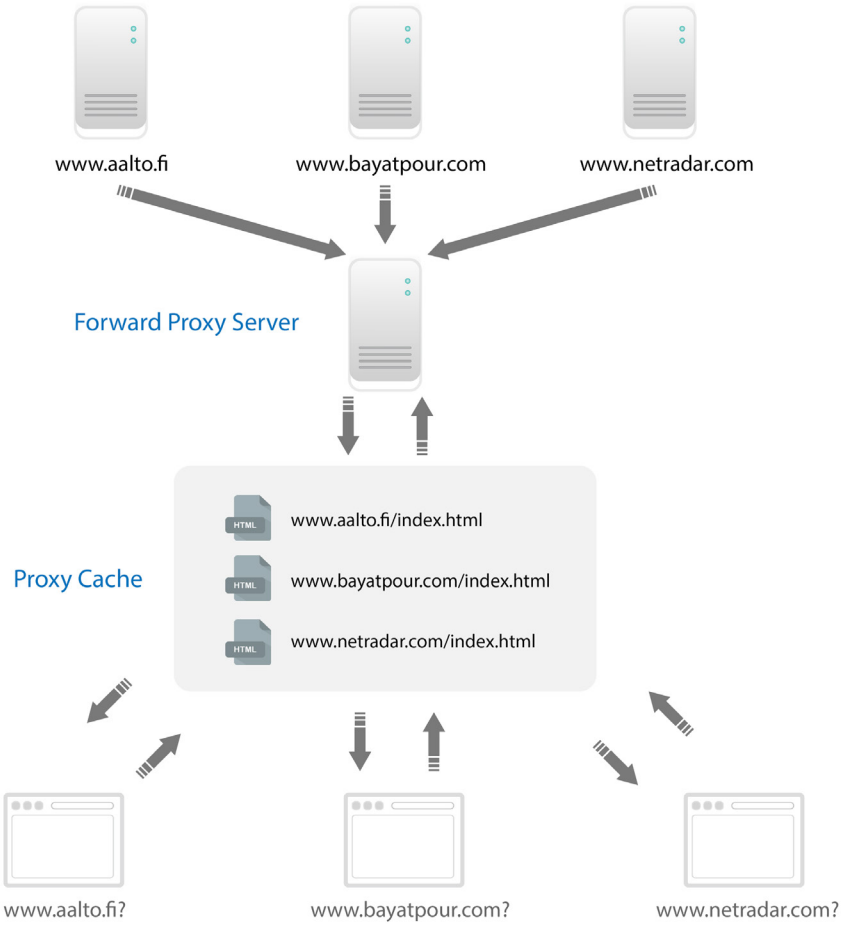

Figure 3: Forward Proxy Caching

2.2.3.2 Reverse Proxy Caching

Like the Forward proxy, a Reverse proxy caches also reside on the edge of a network, but it delivers the content of a finite number of internal documents to the millions of external users on the entire Internet.

Requests for content to an internal Web server are filtered through the proxy cache before they reach the source Web server. Reverse proxy caches, which store the most frequently accessed data, are optimized to serve data quickly and efficient. [27] Figure 4 explains caching in a Reverse proxy.

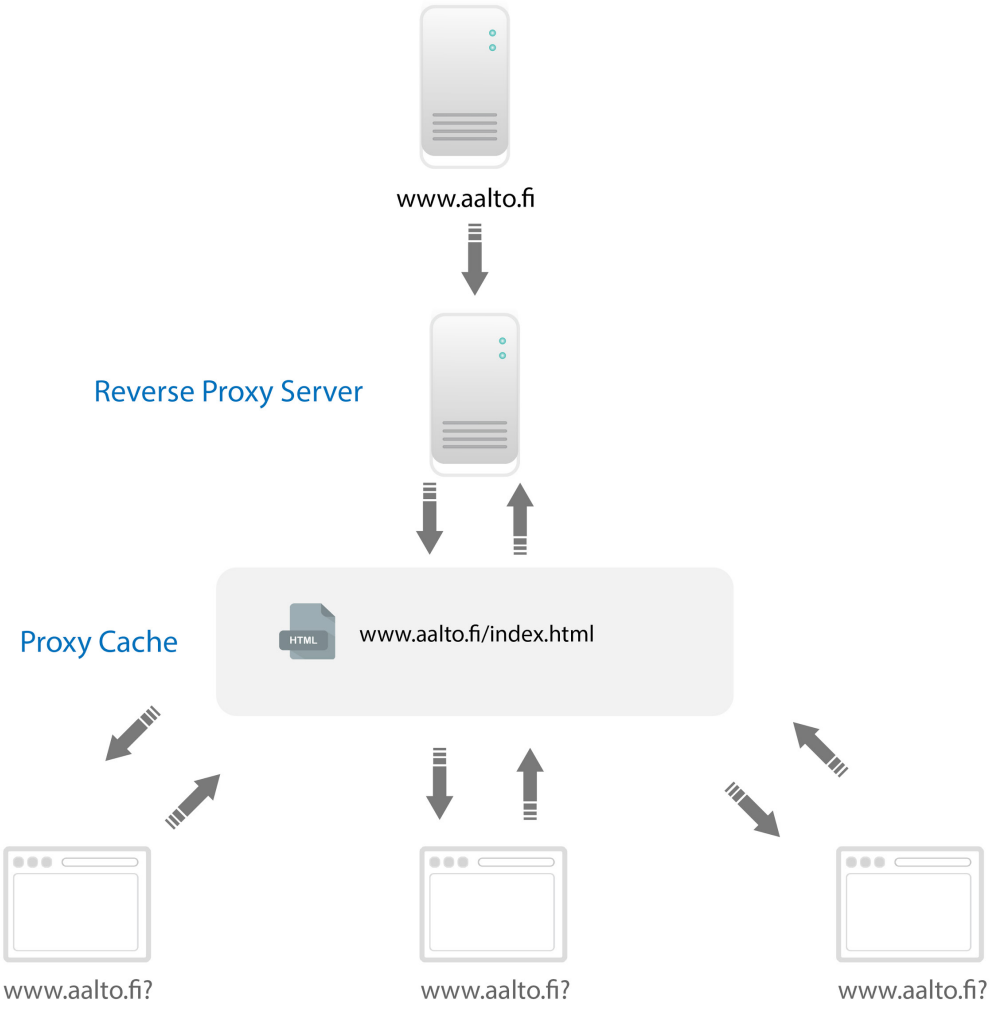

Figure 4: Reverse Proxy Caching

2.2.3.3 Server-Side Dynamic Caching

This caching technique improves performance and scalability by executing only necessary code on the Web application server or in the required database. A specifically designed filter between the server and the clients caches the output of dynamic requests.

Once the HTML has been saved to the Web content cache, the filter detects subsequent requests to the same dynamic page, intercepts the requests, and immediately responds with the cached output. Therefore, the Web server does not try to fulfill the request. This caching process also saves bandwidth and processing cycles because the database does not need to be accessed and no business logic needs to be applied to the database. [27]

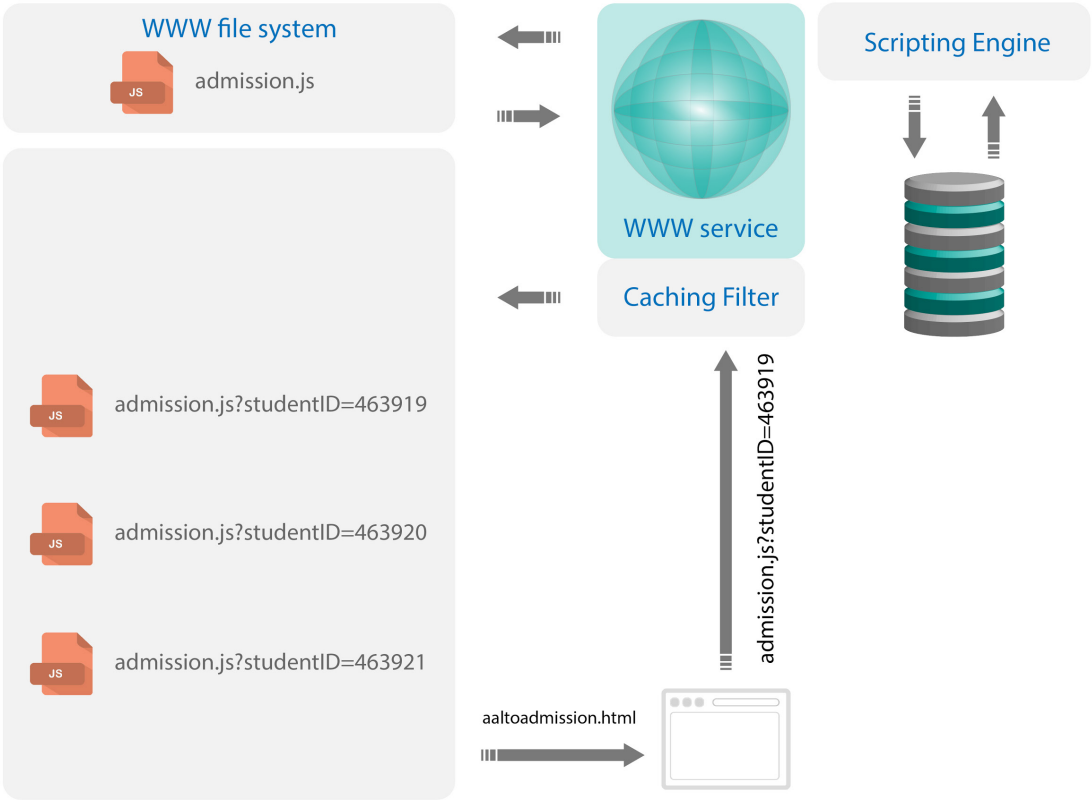

Figure 5: Server Side Dynamic Caching

Some server-side dynamic content caching solutions also implement page compression. This functionality, which is well explained in the next section; removes all the remarks and white space inside the HTML that is sent to the clients. Additionally, the file can be zipped, greatly reducing its size. Such cleaned and compressed pages download much faster and further enhance the user's experience with the site.

2.2.4 Content Delivery Networks

A content delivery network (CDN) is a system of distributed servers that deliver web pages and other Web content to a user, based on the geographic locations of the user. Having more hops between client and the webserver mean more time to render data from a request on the user's browser. The speed of delivery is also constrained by the slowest network in the chain. The solution is a CDN that places servers around the world and, depending on where the end user is located, serves the user with data from the closest or most appropriate server. CDNs reduce the number of hops needed to handle a request. [28]

The CDN copies the pages of a website to a network of servers that are dispersed at geographically different locations, and caches the contents of the page, which are permitted to be cached. When a user requests a webpage that is part of a content delivery network, the CDN distributor or switch will redirect the request from the originating site's server to a server in the CDN that is closest to the user and can deliver the cached content. The CDN will also communicate with the originating server to deliver any content that has not been previously cached. The process of bouncing through a CDN is nearly transparent to the user. The only way a user would know if a CDN has been accessed is if the delivered URL is different than the URL that has been requested.

CDNs generally push the Edge Network closer to end users. The Edge Network grows outward from the origin server by the addition of co-location facilities, bandwidth, and servers. CDNs choose the best location for serving content while optimizing for performance.

They may choose locations that are the fewest hops or fewest number of network seconds away from the requesting client. CDNs choose the least expensive locations while optimizing for cost. [29] CDNs use various techniques such as "Web Caching", "Server Load Balancing", and "Request Routing" to achieve the optimization goals.

- "Web Caches" store popular content closer to the user. These shared network appliances reduce bandwidth requirements, reduce server load, and improve the client response times for content stored in the cache.
- "Server Load Balancing" uses a web switch, content switch, or multilayer switch to share traffic among a number of servers or web caches. In CDNs, the switch is assigned a single virtual IP address. Traffic arriving at the switch is then directed to one of the real web servers attached to the switch. This has the advantages of balancing load, increasing total capacity, improving scalability, and providing increased reliability by redistributing the load of a failed web server and providing server health checks.
- "Request routing" directs client requests to the content source best able to serve the request. This may involve directing a client request to the service node that is closest to the client, or to the node with the most capacity. A variety of algorithms for Global Server Load Balancing (shown in preceding diagram) are used to route the request. Choosing the closest service node is done using a variety of techniques including proactive probing and connection monitoring. [29]

CDN is mostly effective in speeding the delivery of content of websites with high traffic and global reach. The closer the CDN server is to the user geographically, the faster the content will be delivered to the user. CDNs also provide protection from large surges in traffic. Content delivery networks are mostly used for B2B interactions and in serving content to consumers.

Today, as more aspects of daily life go online, organizations use content delivery network to accelerate static content, dynamic content, mobile content, e commerce transactions, video, voice, games and so on. [30] Figure 6 briefly illustrates how a CDN works.

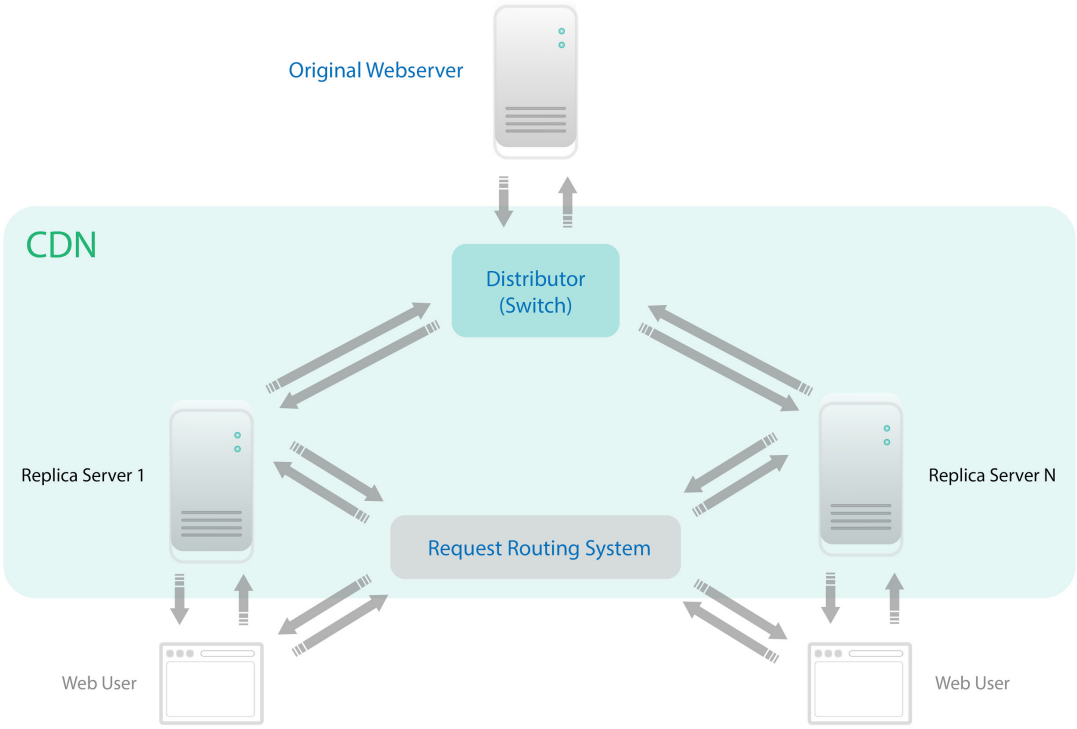

Figure 6: CDN Architecture

2.3 Commercial Solutions

There are always some companies which merge available optimization solutions to offer competitive content delivery services to their clients. In many of the cases they call themselves CDNs who deliver the content while the content is optimized at their proxy servers before delivery to the client. Some of them are free CDNs such as Coral Content Distribution Network, FreeCast, PPLive, QQLive, and PPStream. But some others which are more popular are commercial CDNs such as; Akamai, Amazon CloudFront, Internap, Limelight Networks, CacheFly, Windows Azure, and CloudFlare. Next two sections in this chapter, will introduce two of the most popular CDN solution providers who claim offering novel optimization solutions for web image content type delivery.

2.3.1 Akamai

Akamai is recognized as an old player and global leader in Content Delivery Network services, claiming to make the Internet fast, reliable and secure for its customers. The company claims to offer advanced web performance, mobile performance, cloud security, and media delivery solutions, which can revolutionize the way businesses optimize consumer, enterprise and entertainment experiences for any device, anywhere within the internet. The company operates a network of servers around the world and rents capacity on these servers to customers who want their websites to work faster by distributing content from locations close to the user. Over the years their customers have included Apple, Facebook, Bing, Twitter, eBay and healthcare.gov. When a user navigates to the URL of an Akamai customer, their browser is redirected to one of Akamai's copies of the website. Numbers of optimization categories are offered by Akamai today for web performance, media delivery, cloud security, cloud networking, and CDNs for operators; which all benefit from almost the same collection of solutions but in different business sale packages.

The most interesting solution category of Akamai from this thesis point of view, might be the Akamai "website" and "mobile" performance solutions. Akamai "mobile performance optimization" solutions focus on three key categories of "APIs", "Mobile Apps" and "Mobile Websites".
Akamai's APIs performance solution provides optimization by:

- API Acceleration; which uses Akamai collected real-time data in order to select an optimal path between the origin infrastructure and the Akamai Edge servers.
- API Caching; which caches the API responses at the Edge to position the content closer to the consumer requesting it, while keeping requests off the cellular network to save the battery life of mobile devices.
- API Compression; which compresses API responses in order to reduce the payload size up to 90 % and the delivery time of API responses as well.

Akamai's Mobile Apps performance solution provides optimization by:

- Offering latest performance standards; such as HTTP/2 and IPv6. At Akamai, they perceive enabling secure connections for HTTP/2 as a path to better SEO. They also turn on IPv6 for the mobile app client connections even if the origin infrastructure is not enabled for it.
- Overcoming network uncertainty for mobile apps by utilizing proprietary network detection routines in order to automatically determine if the user is on a good or poor network, and adjust image compression settings in real-time to maximize the user experience. This solution is primarily used in "Mobile Websites Optimization" category which is going to be explained next in this section.

Akamai's Mobile Websites performance solution provides optimization by "Image Converter" in order to enable organizations for dynamically manipulating images in the cloud through appending application programming interface (API) commands to image URLs. Image Converter supports real-time API commands including:

- Downsize: reduces the image's dimensions.
- Resize: scales images to a specific width and height.
- Crop: cuts out a section of an image based on dimension and axis parameters.
- Change Output Quality: compresses JPEG images based on a 1 to 100 scale.
- Change Output Format: changes JPEG, PNG, GIF & TIFF images to a specific file type such as JPEG, PNG & GIF.
- Background Color: sets the background color for transparent images using HTML or Hex colors.
- Compose Images – place an image in a specific location on top of another image e.g. for watermarking. [31]

All the above mentioned solutions are applied in different optimization categories at Akamai to improve QoS, and QoE for different businesses all around the internet. However, the point is that how long some of these solutions may last while the customer attitudes tends to change so often. As an instant imagine the day in which the users want to save images form websites on their mobile phones, and instantly print them. Then the Image Converter solution of Akamai cannot improve the QoE anymore as the users will be unhappy with the missed pixels on the printed images. That is where new lossless image compression methods might be help.

2.3.2 CloudFlare

From an engineer point of view, "CloudFlare" can be seen as a web security company offering security at its Content Delivery Network, while it claims mainly to be a new generation of CDN which optimizes the web content delivery. They offer multiple common solutions for speeding up the Web Content; which are named by them in a way to make a decorative value to popular available web optimization practices. So, they have just added the terminology of web content performance optimization to their business mission as a marketing benefit. Performance Optimization is their marketing statement, not their exclusive solution.

They claim they are a new generation of CDNs because they have powerful data centers and they are able to do some common optimization practices to the web content at their servers before the content delivery to the client for millions of requests at the same time. [32]

In order to gain their maximum claimed performance optimization, the customer needs to implement various number of practices by installing software or configuring source codes. Although they have claimed that there is "nothing to install"; after signing up and trying to really getting it done, the customer will face with numbers of common web content optimization practices which requires either software installation (like "Railgun"; which is used for dynamic pages compression), or code configuration (like "Rocket loader"; which is used to bundle only JS files and storing them on the client's local storage, that requires configuring every single html script tag by customer). All these practices can be also done by the web developers and admins before uploading the content to their web servers. The following figure illustrates how CloudFlare claims to works.

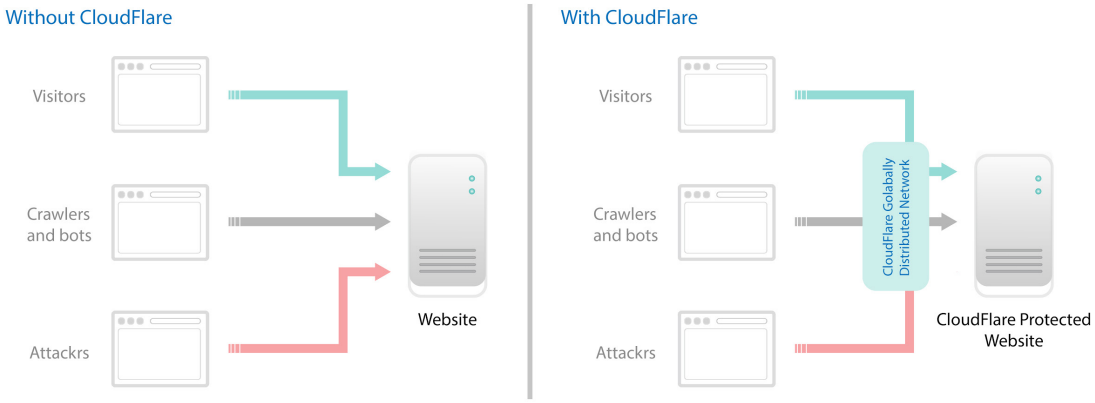

Figure 7. CloudFlare Architecture Comparison [32]

2.4 Vendor-specific Solutions

Vendors who deal with web browsing, always try to improve users' web experience by providing novel optimization solutions. They own sufficient permissions to store and manipulate the data at the client side, in the way the want. So they try to come up with their solutions as browser "extensions" or "add-ons" in order to serve the optimization solution on a platform that they have already owned. In this section two of those vendors and their specific solutions for web content delivery optimization are introduced.

2.4.1 Google Data Saver

The latest Chrome browser can be equipped with Data Saver to significantly reduce data usage by using proxy servers hosted at Google for website content optimization. Google claims that this feature has been shown to reduce the size of web pages by 50 %.

The proxy server receives the request initiated on the client's device, initiates a request for the required resource on behalf of the client, and then optimizes each response before delivering it back to the client. The content optimization is performed by Google's open-source Page Speed libraries, which are specifically tuned for Chrome. The rendering of the page, and all JavaScript execution, is performed by the client's browser. The proxy is equipped with some optimization policies such as:

- HTTP/2: Where possible, the optimization servers request the content via HTTP/2 instead of HTTP.
- Automatic conversion of images: With Data Saver, the optimization server performs an automatic conversion to the new WebP format. WebP is a new image format that supports both lossless (like PNG) and lossy (like JPG) formats. The

automatic conversion to WebP saves an impressive amount of space: on average, lossless images such as PNGs, when converted, became 25% smaller, and lossy images such as JPGs are reduced by up to 34%.

- No images shown at all on slow connections: For slower connections, instead of optimizing the images, Google will opt to send no images at all to the client. Once the page has loaded, it will prompt you with an option to enable images; which it will then fetch, compress and send down to the browser. There's no way to force this by default, so if the client is on a decent 3G connection or even a fast 4G, then there's no option to automatically opt to do this and it's up to Google's discretion, it seems.
- Minification and Compression: Google's server will go through all CSS, JS and HTML content and automatically remove all whitespace to cut down on size. It also ensures that all content is served with GZIP compression. [33]

Although Google has implemented so many optimization techniques to improve the web experience, but there is also another success factor for google, and that is most of websites with high hit rates are hosted or owned by google itself; which will bring more benefits in terms of speed to Google Data Saver.

2.4.2 Opera Mini

Originally, Opera Mini is a web browser designed primarily for mobile phones, and smartphones. Opera Mini requests web pages through Opera Software's servers, which process and compress them before sending them to the mobile phone, in order to speed up the transfer process and reduce the amount of data transferred.

Just like Google Data Saver, Opera Mini tries to save the data and transfer less. The functionality of Opera Mini mode is somewhat different from that of a conventional Web browser, with the amount of data which has to be transferred much reduced, but with some loss to functionality Unlike straightforward web browsers, Opera Mini fetches all content through a proxy server and reformats web pages into a format more suitable for small screens. A page is compressed, then delivered to the phone in a markup language called OBML (Opera Binary Markup Language), which Opera Mini can interpret. The data compression makes transfer time about two to three times faster, and the pre-processing improves the display of web pages not designed for small screens. [34] By default, Opera Mini opens one connection to the proxy servers, which it keeps open and re-uses as required. This improves transfer speed and enables the servers to quickly synchronize changes to bookmarks stored in Opera Link. The Opera Software company maintains over 100 proxy servers to handle Opera Mini traffic. They run Linux and "are massively parallel and massively redundant."[35]

2.5 Academic Ideas

All standard, commercial, or vendor-specific solution got their ideas form researches which are mostly held at academic institutes. Taking a look at some of the academic ideas for web content optimization might familiarize us with the future or upcoming trends in this field. In the next sections two academic ideas which have aspects similar to the proposed IUC method in this thesis, are explained.

2.5.1 Jukka Manner and Le Wang "Web in a Bundle"

One of the most recent and relevant research works in HTTP request minimization might be the 2013 "Jukka Manner" and "Le Wang" bundling proxy system; in which they proposed a method for reducing the number of HTTP requests and consequently achieving webpage loading time acceleration by utilizing an HTTP proxy server. This bundling proxy requests a webpage and all its linked assets on behalf of a client from a web-server, and then rewrites the URLs inside the page for their new resource destination on the client's (browser) Local Storage.

Afterwards, the proxy bundles the webpage and all its linked assets in one big JavaScript Blob, and sends it to the client. At client side, the JS Blob will be unbundled, and the "File System API" will be used for handling the files on the Local Storage. Challenges with this method might be shortlisted as; Client's Local storage size limitation, Time to First Byte big delay, Browser Caching disability, and File System API discontinuity from 2014 based on the W3C decision.

As the authors of this research work have mentioned in their paper, [36] their bundling method was mainly aimed to optimize the Mobile Devices Energy Efficiency, while it can remarkably be used in Web Content Speed Delivery Optimization. Their results and test proves up to 39% faster page load time on WLAN Networks, and 20% faster page load time on 3G Networks. Figure 8 shows their EEP (Energy Efficient) proxy structure.

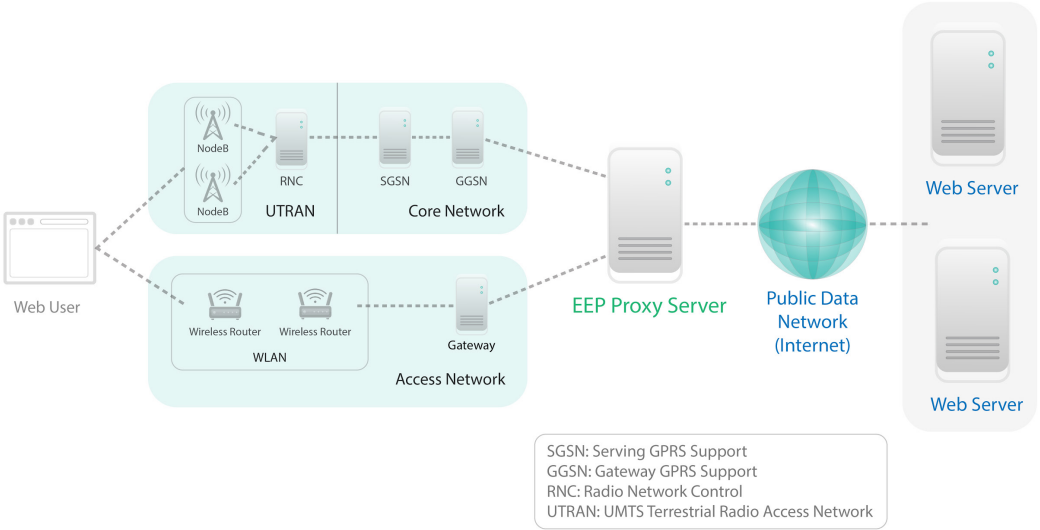

Figure 8: Jukka Manner and Le Wang EEP Bundling Proxy Architecture [30]

2.5.2 MIT JavaScript-based Polaris

Polaris is a new JavaScript framework that aims to shrink webpages load time by 34% at the median. Developed by researchers from MIT's Computer Science and Artificial Intelligence Laboratory and Harvard University, Polaris focuses on reducing latency associated to network trips. Their new approach, is based on two tools:

- Scout, which is able to track the fine-grained data flows across the JavaScript heap and the DOM that arise during the webpage page's load process.
- Polaris, a JavaScript client-side scheduler which leverages Scout graphs to assemble a page.

Dependency analysis is a technique commonly used by browsers to optimize the way they load resources. Before Scout, though, says Netravali, that kind of analysis was carried through based on lexical relationships between HTML tags, which missed many fine-grained dependencies, as it can be seen in the picture below for a real case.

By providing a finer-grained dependency graph, Scout makes it possible for browsers to better schedule resource loading, and this is where the Polaris JavaScript scheduler comes into play. Polaris claims that can run unmodified on common browsers and is able to calculate the dynamic critical load path for a page based on its Scout graph, which the server can bundle with the HTML page it serves together with Polaris itself. The dynamic critical load path, which by definition is the path which currently has the most unresolved objects, is different than the static load path as provided by Scout in that it is influenced by the order and latency with which network fetches complete.

Polaris prioritizes the fetching and evaluation of objects along the dynamic critical path, trying to make parallel use of the client's CPU and network, and trying to keep the client's network pipe full, given browser constraints on the maximum number of simultaneous network requests per origin.

The researchers behind Polaris tested their system under a range of network conditions, "with latencies ranging from 25ms to 500ms, and bandwidths ranging from 1Mbps to 25Mbps", and on 200 popular websites. This showed, they say, a decrease by up to 34 percent at the median and 50% at the 95th percentile. Performance varied significantly across sites, being higher with complex pages and lower with pages making aggressive use of caching. [37]

As it was discussed and explained in this chapter, there are plenty of available optimization solutions which are all beneficial in their specific own scale. However, all of them has their own requirements and limitations. At the beginning of the chapter, standard solutions such as GZIP and DEFLATE Compression schemes were explained which required being enabled at webservers and also being supported by client. In addition to these two algorithms, two protocols of SPYDY and HTTP2 were also explained, which had their secure connection limitation and single webserver multiplexing issue. Afterwards in this chapter, other standard and mostly hardware

solutions such as “server-side caching” and “content delivery networks” were introduced; which have their own hardware and processing power limitations and requirement to achieve the highest possible benefits. Afterwards we explained some commercial solutions form Akamai and Cloud Flare in order to explain how companies merge and offer different solutions at the same time on their servers in order to make their content delivery service competitive enough, while they might require the web owner to do numbers of different manual setting in order to achieve the highest possible benefits. Also vender-specific solutions such as Google Data Saver and Opera Mini were discussed in order to show how different web browser developers, use their platforms and proxy servers all around the globe to make benefits from their optimization solutions which mainly are based on lossy compression of the data.

Finally, two of the most recent and novel academic ideas towards web content delivery optimization were explained, which showed the potential of other solutions towards content delivery optimization in near future. Although, those academic ideas have their own limitations such as; the “File System API” restriction on “Web in a Bundle” method, and “1-26 Mbps connection speed” limitation for Polaris, with the advanced in web technologies, there might be less barriers in future for these such academic ideas to achieve more benefits. [37]

3 Research Methodology, and Experimental Setup

The proposed method and system in this thesis is designed with the intention of bringing benefits both to the clients (end users) and also data owners (CDNs, web owners, web developers, and etc.) in terms of reducing number of HTTP requests and also increasing the efficiency of HTTP compression; while keeping the render quality original and lossless.

Involved parties might benefit from web performance optimization, data transfer size reduction, page loading acceleration, and resource/power consumption optimization. The whole idea might be briefed as encoding images into Base64 format and bundling them inside a JS file in order to reduce the number of requests and also make benefit out of the proposed Aggregative Compression for optimizing the data transfer efficiency. As the initial step to explain the methodology; an in depth explanation about Base64 could broaden the

3.1 Base64

At a young age, we learn to count on our fingers - starting out with 1-5, then 1-10, and maybe, if you're particularly enterprising as a toddler, you will learn to count to 20, 30, and beyond. No one ever attempts to enlighten us that we are actually making some more complex mathematical assumptions; we all know Base10, to be precise. So, why did we choose Base10? It's not because the letterforms 0-9 exist; that was actually a result of the choice to use Base10.

In fact, it is most likely because of the learning process we decided above - we have 10 fingers. This makes it much easier to understand the system. Consider the number 1020. Starting from the right, we can understand this as "0*1 + 2*10 + 0*100 + 1*1000". Now, consider the number 5,378. We can understand this as "8*1 + 7*10 + 3*100 + 5*1000". We can see that there this is a generalizable formula for understanding all base systems.

Base2, or binary consists of two digits, 0 and 1. This lends itself well to computing for many reasons, most fundamentally because computers rely on switches that have two states: on or off. Binary is the most basic system needed for all logical operations (think "true" and "false"). "If binary is all that computers are made of; how would a user write letters in binary?" This actually brings us to Base16. Also known as hexadecimal, Base16 uses the numbers 0-9 followed by the letters a-f (not case-sensitive). In particular, you will see hexadecimals used to define RGB colors in CSS (and in most color-picker widgets on desktop software), with two digits for each of the channels red, green, and blue.

So, for instance, #A79104 would produce r = A7, g = 91, b = 04. In decimals, this would be equivalent to r = 167, g = 145, b = 4; the resulting color would be a golden yellow. Two hexadecimal digits put together can represent 256 different numbers, and thus there are 256^3 (16,777,216) possible number combinations in the RGB hexadecimal system, represented by only 6 characters (or 3 if you use the shortcut method, where each of three

digits is implicitly doubled; e.g. #37d == #3377dd). Base16 is often used in assembly languages, which is the lowest level accessible programming language.

Base32 definition is actually an encoding that starts with the first 26 letters of the alphabet and ends with the numbers 2-7. This is defined in the Internet Engineering Task Force's Request for Comments (RCFC) 4648, which also defines Base16 and Base64. Note, the difference is that the encoding for 0 is A, not 0. To encode a string in Base32, the following instructions happen. First, the string to be encoded is split into 5 byte blocks (40 bits in binary). Letters are represented by 8 bit blocks in ASCII (the standard for computers), so for every 5 letters, there are 40 bits. (This 8-bit definition for each letter allows for a total of 255 characters in ASCII.)

Base64 follows a similar process. There are a few fundamental differences between Base32 and Base64. Base64 includes the letters A-Z, a-z, numbers 0-9, and the symbols + and /. As mentioned previously, the "=" symbol is used for padding. The differences are mainly that all letters are case-sensitive, and all digits are used (instead of the subset 2-7). The symbols + and / are also added. The Base64 encoding process takes 24-bit strings (3 letters) and breaks them into four 6-bit chunks, mapping the resulting binary number to the Base64 alphabet. Next, divide these 40 bits into 8 five-bit blocks; so, for every 5 letters, there are 8 blocks to encode in base32. Map each of these blocks to a 5-bit character mapping in the Base32 alphabet. For instance, if the five bit block is 00010 (or decimal 2), the mapped character is the letter, c. If the five bit block is 01010 (decimal 10), the mapped character is the letter K.

Nowadays Base64 is a generic term for a number of similar encoding schemes that encode binary data by treating it numerically and translating it into a base 64 representation. The Base64 term originates from a specific MIME content transfer encoding. Base64 encoding schemes are commonly used when there is a need to encode binary data that needs be stored and transferred over media that are designed to deal with textual data. This is to ensure that the data remains intact without modification during transport. Base64 is used commonly in a number of applications including email via MIME, and storing complex data in XML.

The only downside is that base64 encoding will require around 33 % more space than regular strings. So with base64 anyone can encode and transfer any sets of binary data through any system and then decode them to original binary data. Anyone can also build her/his own base64 algorithm for her/his specific needs. [38]

The very important point about Base32 and Base64 is that, not only there is no downside when encoding numbers with them, but also there is a remarkable character saving. In today's web, in addition to using hexadecimal numbers on a regular basis for CSS colors; Base32 and Base64 are used on the web consistently.

Although the official encoding process for Base32 and Base64 bloat the size of the string, encoding numbers in Base64 or Base32 can be very beneficial for things like URL shortening. Consider the following decimal numbers and their Base32 equivalents.

Decimal	Base16	Base32
20	U	U
50	bs	y
967	6h	PH
745619	WYET	C2CT
7241930	G5AGK	boDK
798312345192	xhpr7lti	LnfH65o

Table 2: Base32 benefits on number shortening

As it can be conceived from table 2, above table, there are significant advantages to using Base64 or Base32 for number shortening. When every character counts, using these base encodings allows you to save characters. In many cases, the encoded number is about half the length of the non-encoded number.

3.2 Aggregative HTTP Compression of Images in a JS Bundle

It might be a big deal to compress images on the web in a lossless manner, because they are mostly already compressed by their original format. Now we know how Base64 encoding can convert any binary data into textual format, and we know how HTTP compression can be more efficient on longer textual contents by having a higher chance of more repetitive characters occurrence. Therefore, this could be the time to combine these two features in order to obtain more efficiency on Image Content Type Compression over the web. The proposed method in this thesis uses a novel technique which converts all the images that are included in a webpage into textual format using Base64 encoding, and then puts them as different arrays of a JS object inside a JSON file; called the “JS Bundle” of images.

The point is that the first encoded image at the top of the JS file will get the same compression rate compared to the case that it was not included in a JS file including all other images, but the second image will start to get more compression rate as the first characters of the second image textual data have this opportunity to be found repetitive and considered as pointers instead of characters. This procedure goes on and brings compression benefits to the last image textual content inside the JS Bundle.

3.3 Methodology Description

The idea of the proposed web content delivery optimization method in this thesis comes from aggregating “textual images content” in a JS bundle; which leads to a cut down in number of requests from client to multiple web servers, and also results in compression efficiency improvement. Implementing the proposed method as a proxy that is called “IUC” (Image Unifier and Compressor), improves the “user” web experience in terms of speed; and at the other side of the game, enhances the performance of “webpages”.

When IUC proxy is deployed as a “B2C” service, it brings benefits to the “users”; having a faster Web Experience and also less data transfer costs; while keeping the render quality as original. Those who are granted an IUC username and password, can easily browse their desired web pages through the “IUC Proxy Server”. When IUC is deployed as a “B2B” service, it brings benefit to the web servers’ CDNs in terms of transfer size and transfer time, and at the same time it improves their “webpages” performance at the client side; which will might also bring them a remarkable SEO advantage on the web among other webpages which are not equipped with IUC technology. The SEO effects of the proposed method used in CDNs is discussed in later sections.
Proposed method and developed IUC Proxy Server comprises:

1. The IUC proxy receives a request for a webpage from the client.
2. The proxy fetches the web page from the web-server.
3. The proxy decompresses the webpage, if it’s compressed at the web-server.
4. The proxy fetches all the webpage linked assets including images, and analyzes the fetched images, whether they fit into the bundling policy (image size limit) or not.
5. The proxy equips the webpage with the IUC Cache JS function, and modifies image tags inside the webpage with IUC specific identifier; that handles rendering and also caching of the images.
6. The proxy sends back the compressed (using the client’s supported compression schemes such as; GZIP, Deflate, etc.) and modified web page to the client; which might even happen before receiving all the linked assets of the webpage from the web server.
7. The proxy compresses all other fetched assets than images; which are linked inside the webpage; using the client supported compression scheme (GZIP, Deflate, etc.).
8. The proxy sends all other linked assets of the webpage than images to the client.
9. The proxy encodes those images that fit the bundling policy into Base64 format. (The bundling policy in developed IUC proxy is to encode only images with the maximum size of 40 KB)
10. The proxy puts them inside a JS file; as objects of an array.
11. The proxy updates the proxy state about the bundled images info.
12. The proxy compresses the JS bundle of base64 images in order to make benefit out of the IUC Aggregate Compression; using the client’s supported compression schemes (GZIP, Deflate, etc.).
13. The proxy sends the images JS bundle to the client.
14. Then at the client, the IUC JS caching function and file; which was added at proxy to the bottom of the webpage; will handle reading and rendering the base64 image data from the JS bundle to the browser window.
15. If the client requests another page; which includes any of the previously encoded fetched assets; then the proxy knows that from its state database and modifies the IUC Cache JS function to render those images from the available JS bundle at the browser cache, and new images which fits into the bundling size policy will be again bundled in inside a JS file.

The Following diagrams shows how in IUC proxy, the flow is designed.

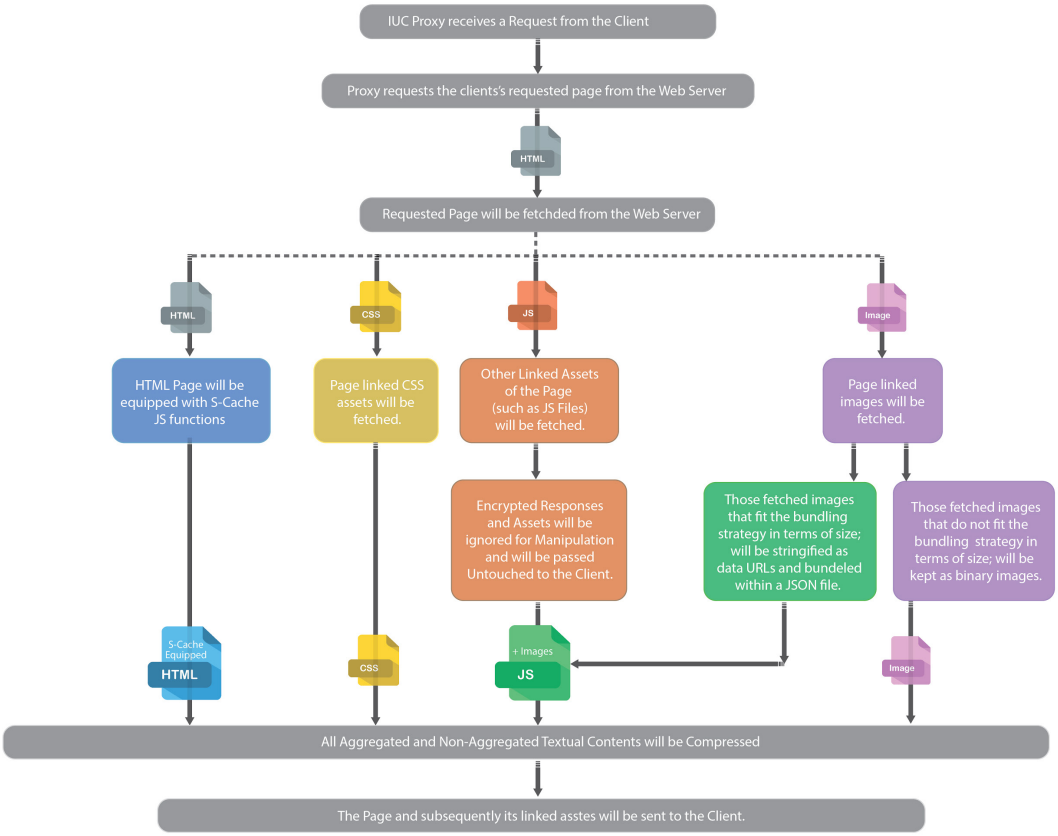

Figure 9: IUC Proxy Flow Design

3.4 Experimental Setup

The proxy is setup as a middle box between client and server in order to handle “HTTP Requests Minimization” and “Aggregative HTTP Compression” for the whole content delivery optimization service. Nowadays most of the websites with high hit rate include so many 3rd party advertisements which make the page size and number of http requests to be different in each page load experience on the same page. Because of the mentioned web advertisements issue, and also in order to make a reliable and static testing condition for defining the maximum potential of the technology benefits; a test web shop is developed and uploaded to a webserver in UK. This test web shop also calculates and reports the page load time with its most important details in order to make it possible even for mobile users to easily see the test results without the need of having the “developers’ tools” on their mobile browsers. The test web shop has also the capability to be loaded with 3 different weights. The light weighted web shop includes 50 images, the average weighted web shop includes 100 images, and the heavy weighted one includes 150 images. These tests on each “test web shop” page weight category are done through:

- Fast Wi-Fi connection (with download speed of 150 Mbps) with stable bandwidth and traffic, on a MacBook Pro in Safari 9.1.2 (11601.7.7), Mozilla Firefox “45.2.0”, and Chrome “51.0.2704.103”
- Gigabit Ethernet connection (with download speed of 848 Mbps) with stable bandwidth and traffic, on a OS X EI Capitan MacBook Pro and Windows7 PC in Safari 9.1.2 (11601.7.7), Mozilla Firefox “45.2.0”, Chrome “51.0.2704.103”, and IE “11.0.9600.18349”.

After all the above optimistic tests, real world wide web tests conducted in order to see the practical benefits from the user experience perspective. For this reason, from Finland top 100 Alexa site, “yle.fi” with the rank of 8, was selected. The site has the minimum amount of single page size variation which provides stability in data transfer size. Also “yle.fi” has the “Lazy Loading” feature which gives the page this ability not to request and load those images that are located in non-scrolled down part of the page. This way the browser will request and load only the images that are located at the top currently viewable part of the page, which could be a challenging issue for the IUC Proxy as the proxy can unify and compress less images in a single JS bundle. These tests on “yle.fi” home page are done through:

- 2G bandwidth (with download speed of 0.2 Mbps) provided by an iPhone 5s Hotspot via Wi-Fi to a MacBook Pro with OS X EI Capitan in Chrome “51.0.2704.103”.
- 3G bandwidth (with download speed of 15 Mbps) provided by an iPhone 5s Hotspot via Wi-Fi to a MacBook Pro with OS X EI Capitan in Chrome “51.0.2704.103”.
- 4G bandwidth (with download speed of 32 Mbps) provided by an iPhone 5s Hotspot via Wi-Fi to a MacBook Pro with OS X EI Capitan in Chrome “51.0.2704.103”.

Beside all these tests, one Squid proxy which benefits from proxy caching, has been also implemented on the same server machine but at a different port, in order to clarify how much of the gained benefits are from the proxy caching, or even how much the Bundling process can cost the whole load time or harm the proxy caching benefits. The following diagram shows the Experimental setup and the workflow of handling HTTP Requests Minimization.

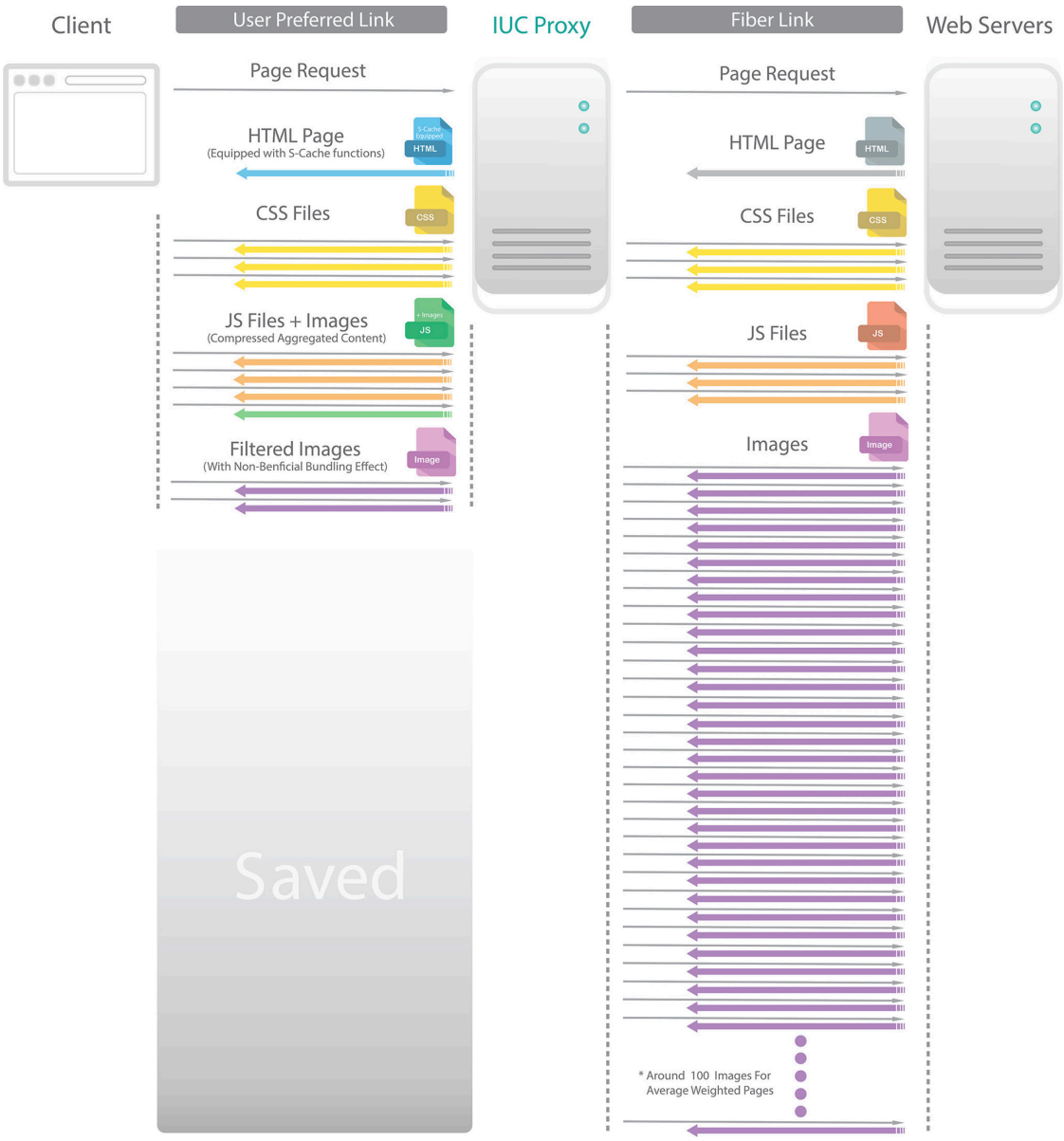

Figure 10. HTTP Requests Minimization by IUC Proxy

As it's also drawn in the Figure 10, the proxy equips the HTML page; linking to the bundled images, with proxy assigned "IDs" and "Remote Cache Function" for images, to handle the tracking of the bundled images and ensuring the maintenance of the client side caching benefits.

If any compression scheme is supported by client; then HTTP compression is also applied to all the content before delivering them to the client. The following diagram shows the Experimental setup and the workflow of handling the Aggregative HTTP Compression in IUC Proxy.

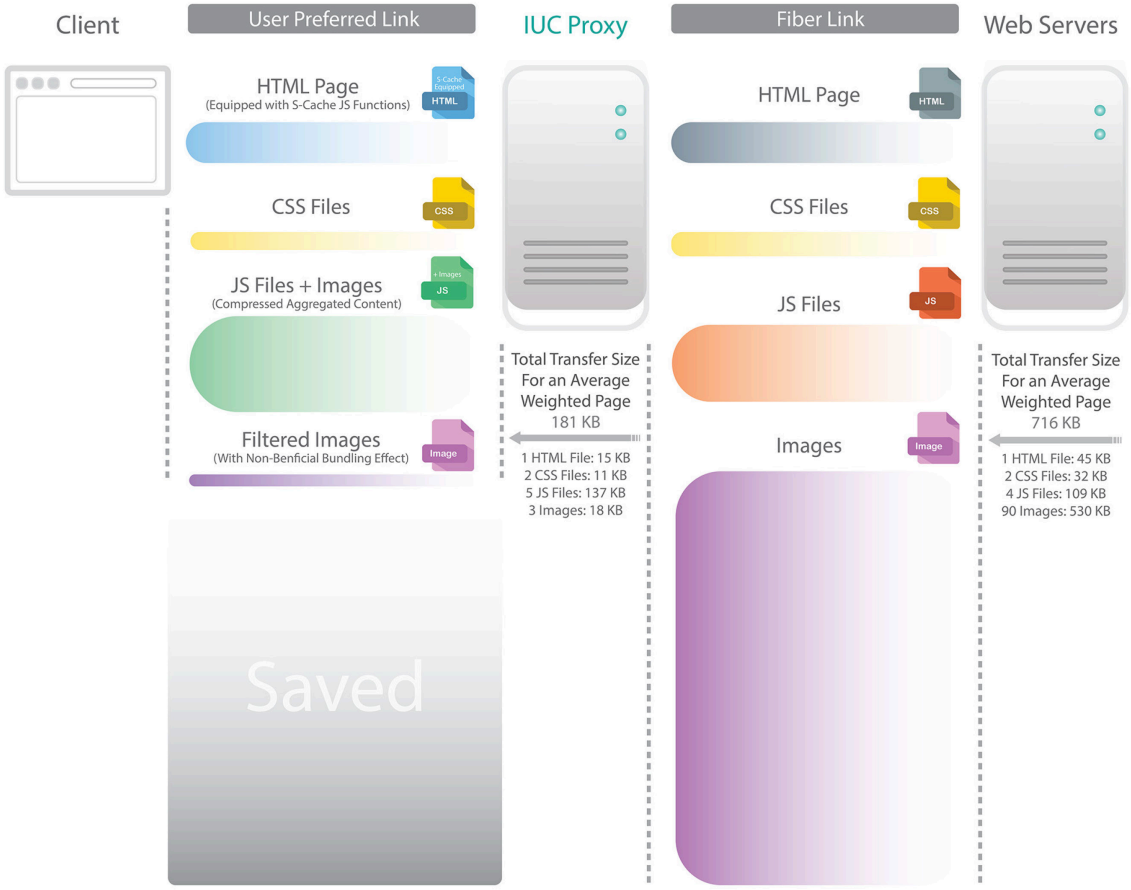

Figure 11: Aggregative HTTP Compression by IUC Proxy

4 Measurement Results, and Analysis

In this chapter the results of all mentioned tests in the previous section; are going to be reported and analyzed. The reliability of the results is also examined through practical world wide web situation loading tests. The conducted tests with 2G, 3G, and 4G bandwidths are also intended for Mobile QoS optimization tests and analysis.

4.1 Wi-Fi Connection Test and Analysis

In this section, the results of the tests through a fast Wi-Fi connection (with download speed of 150 Mbps) on OS X EI Capitan in different browsers are reported and analyzed. The reported loading time results are the average amounts of gathered values from 25 repetitions per each test Setup.

The first test result chart illustrates how IUC Proxy brings benefits to the test web shop page load time in Chrome. It can be derived from the chart that as the page size increases, the efficiency of IUC proxy in decreasing the page load time also improves; which mainly comes from higher relative elimination of HTTP requests and more Aggregative Compression ratio. Unsurprisingly, IUC proxy decreases the page load time up to 65 % for the heavy test web page, while surprisingly Squid proxy not only didn't decrease the loading time but also increased it up to 54 %.

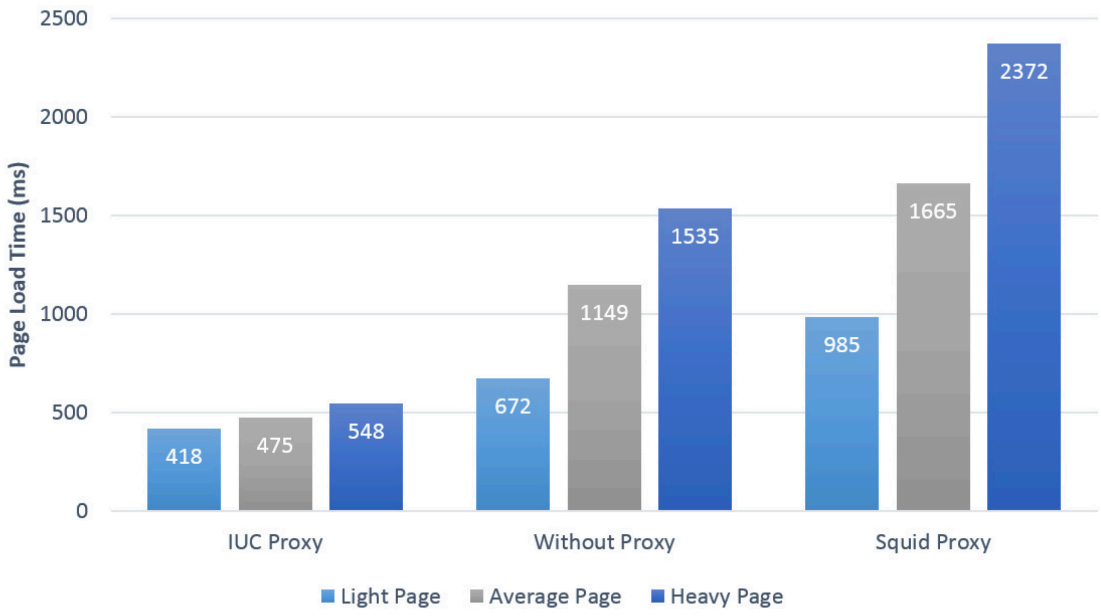

Figure 12: Loading Time results via Wi-Fi on OS X EI Capitan in Chrome

In the next setup, Mozilla Firefox browser is used to see how a different browser on the same operating system and via the same connection type, will behave. The results on Figure 13, show that IUC proxy decreases the page load time up to 60 % and surprisingly

Squid proxy also decreases the page load time remarkably up to 69 %, which means how well the squid mechanism is matched with Firefox browser engine.

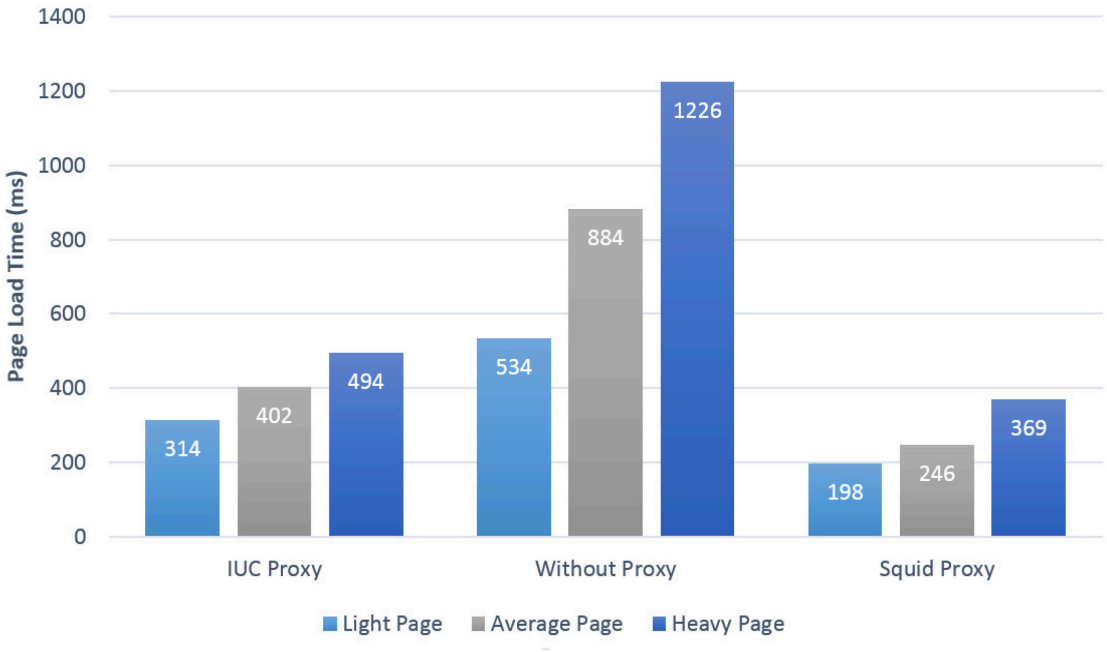

Figure 13: Loading Time results via Wi-Fi on OS X EI Capitan in Firefox

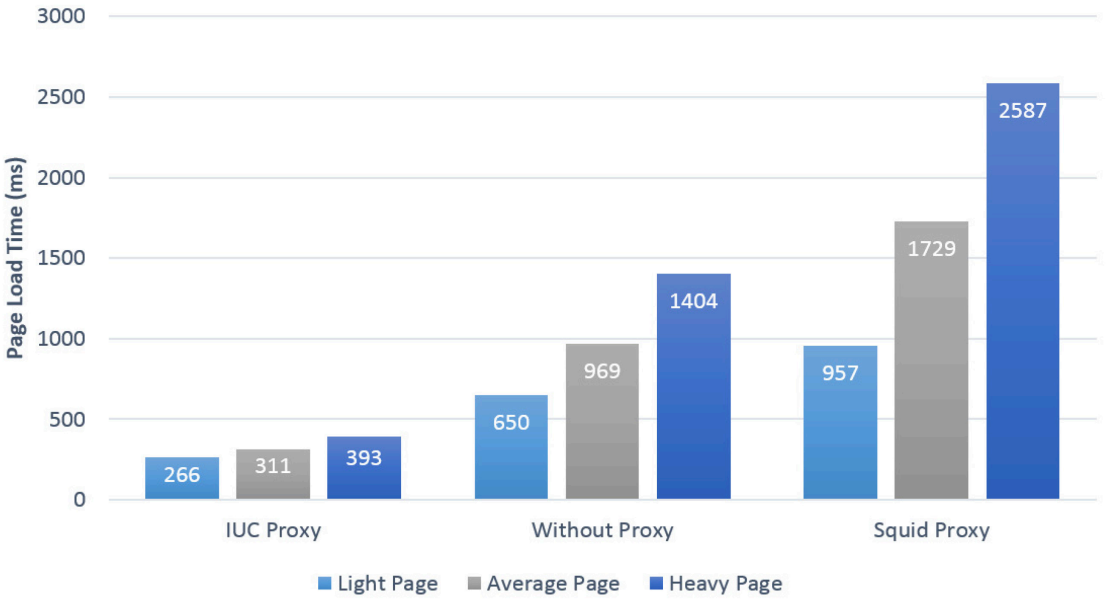

Figure 14: Loading Time results via Wi-Fi on OS X EI Capitan in Safari

Figure 14 illustrates the results being achieved in Safari Browser. Figure 14 shows that the average loading time is decreased remarkably by IUC proxy up to 73 % for the heavy web shop test page, while again surprisingly the Squid proxy not only didn't decrease the loading time, but also drastically increased it up to 84 %, which approximately means doubling the page load time.

It can be derived from the charts in this section that IUC proxy remarkably decreases the page load in all browsers via Wi-Fi Connection on OS X EI Capitan, and performs best in Safari by 73 % faster page load time, but Squid proxy can only decrease the page load time in Firefox, and surprisingly in other browsers it increases the page load time drastically.

4.2 Ethernet Connection Test and Analysis

In this section, the results of the tests through Gigabit Ethernet connection (with download speed of 848 Mbps) on OS X EI Capitan in different browsers are reported and analyzed. The reported loading time results are the average amounts of gathered values from 25 repetitions per each test Setup.

The first chart in this section shows the test results achieved in Chrome. Figure 15, illustrates IUC proxy decreases the page load time up to 65 % for the heavy web shop test page, while Squid proxy not only didn't increase the page load time, but also increased it up to 81 % for the heavy webpage.

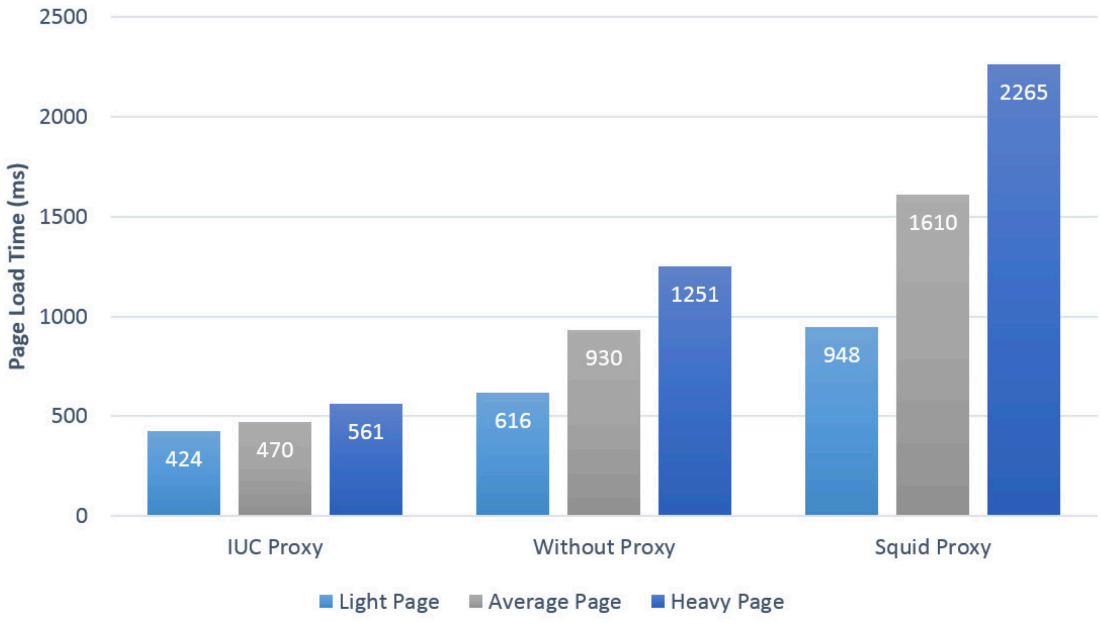

Figure 15: Loading Time results via Ethernet on OS X EI Capitan in Chrome

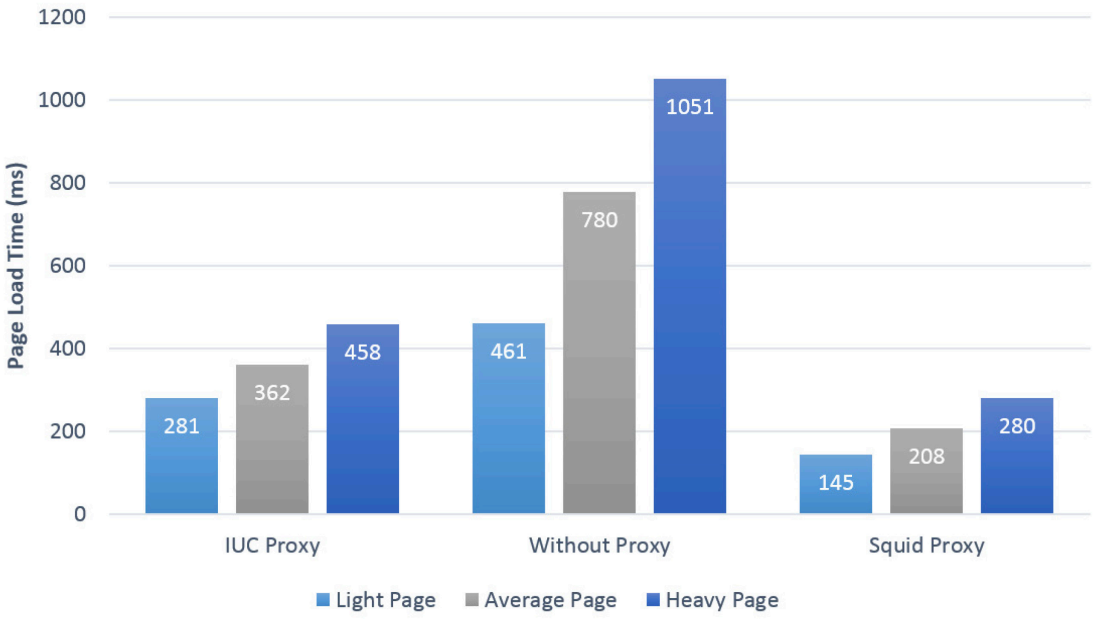

Figure 16: Loading Time results via Ethernet on OS X EI Capitan in Firefox

With regard to the Figure 16, again IUC proxy has decreased the page load time up to 57 %, and Squid proxy also decreased the page load time by 73 %, which makes this assumption that Squid is well optimized for Firefox; more close to be proven.

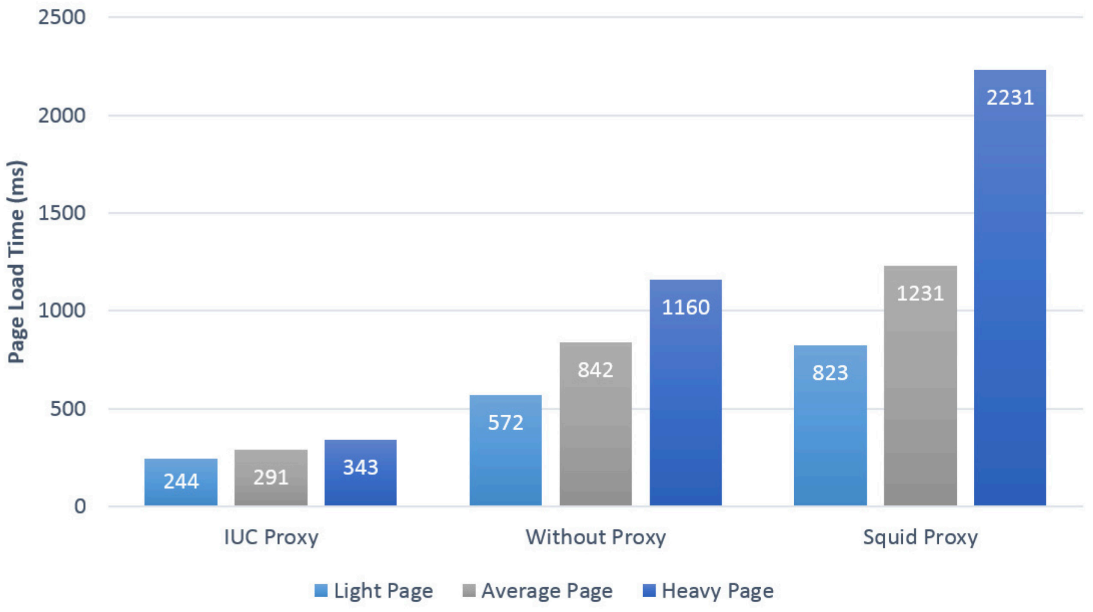

Figure 17: Loading Time results via Ethernet on OS X EI Capitan in Safari

With regard to Figure 17, it can be derived that IUC proxy as a constant trend, again has decreased the page load time drastically up to 71 % for the heavy web page, but on the other side, Squid proxy not only didn't decrease the loading time, but also increased it remarkably up to 92 % for the heavy webpage which is around 2 times more loading time.

Comparing the achieved results via Ethernet and Wi-Fi connections, shows that the results and trends in both connection types are pretty close to each other. In all the tests, IUC proxy remarkably decreased the page load time from 57% to 71%, while Squid Proxy could only decrease the page load time in Firefox.

4.3 Cross Platform Test and Analysis

In order to see how changing the operating system will affect the test result, the previous Ethernet tests are also done on a Windows 7 PC. In the following, the results of the tests through Ethernet connection on Win7 Enterprise in different browsers are reported and analyzed. The reported loading time results are the average amounts of gathered values from 25 repetitions per each test Setup.

Figure 18, illustrates that also on Windows 7 and in Chrome browser, IUC proxy can decrease the web shop page loading time up to 81 % for the heavy weight page. On the other hand, as it might be predictable, just like how it performed on OS X, here also on Win7, Squid proxy not only didn't decrease the page load time, but also increased it by 4 %. The point is that Squid proxy on Win7 harms the page loading time less compared to its behavior on OS X.

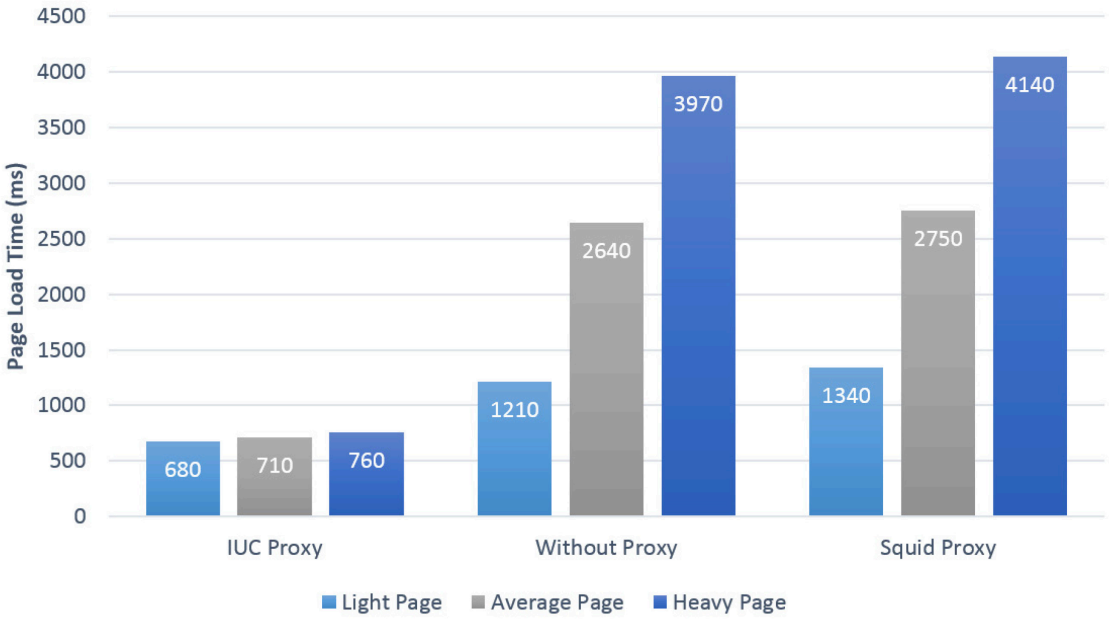

Figure 18: Loading Time results via Ethernet on Win7 in Chrome

The following chart illustrates the results, being achieved in Firefox the Gigabit Ethernet on Win7.

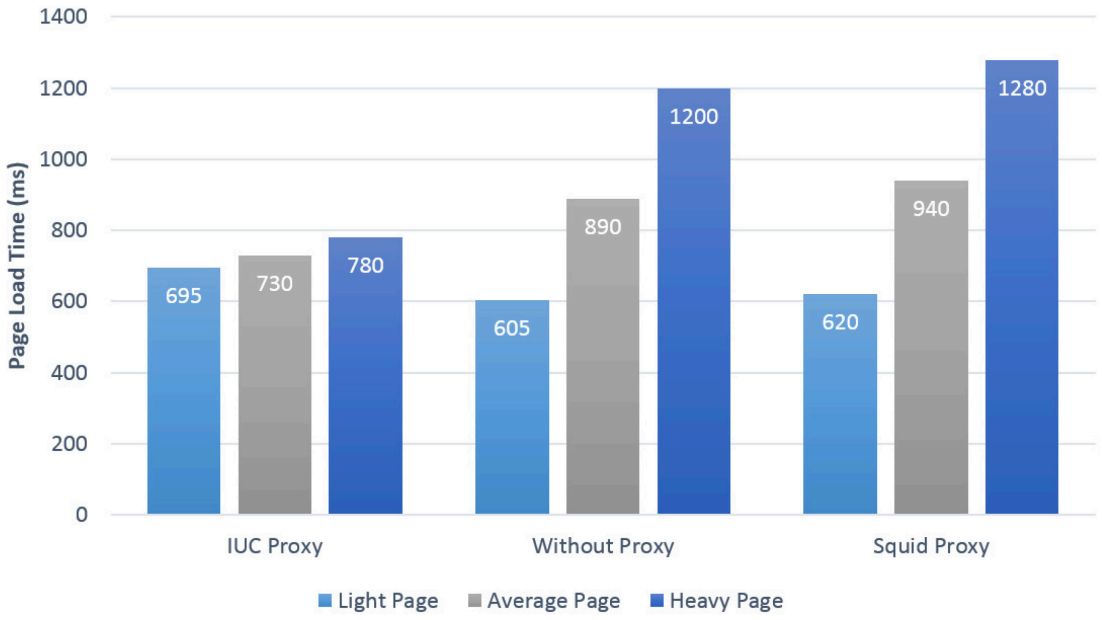

Figure 19: Loading Time results via Ethernet on Win7 in Firefox

Figure 19 shows that IUC proxy is still decreasing the page load time up to 35 % on the heavy page, but not on the light page. Surprisingly on the other side, the Squid proxy which had a good record on OS X in Firefox, but here in Win7 not only couldn't decrease the page load time, but also increased in in all page weights. So the "Being Optimized for Firefox" assumption is rejected with regards to this results for Squid proxy.

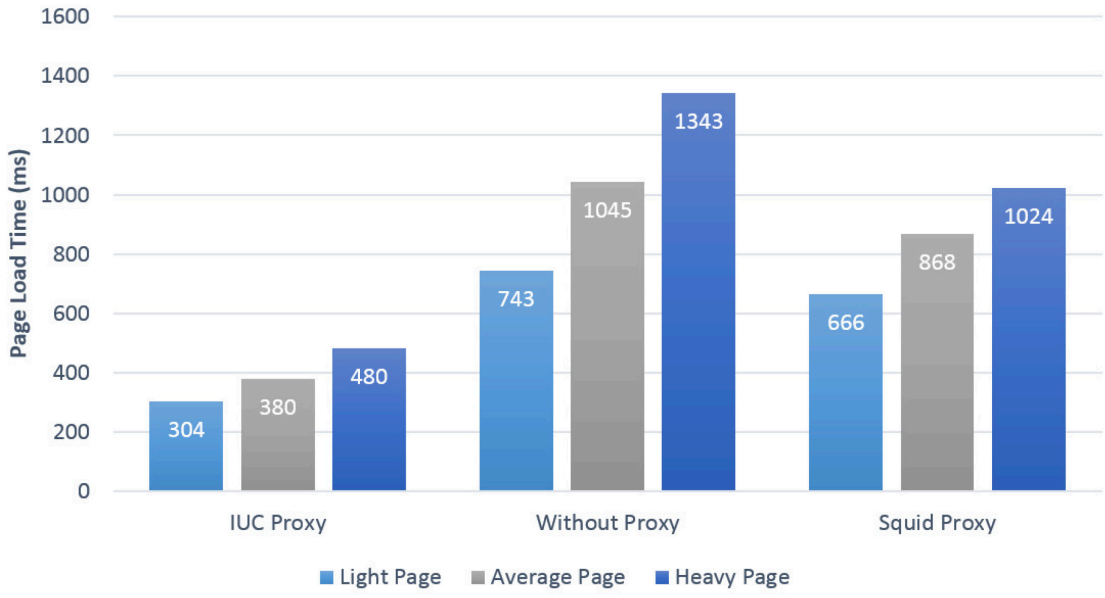

Figure 20: Loading Time results via Ethernet on Win7 in IE

As the default browser on Windows operating system is IE, so in this round of the tests, instead of Safari, the experiment is done in IE. Figure 20 shows that IUC proxy decreases the page load time also in IE up to 64 % and on the other side, Squid proxy also can decrease the page load time up to 24 %.

Comparing the test results on “Win7 Enterprise” and “OS X EI Capitan”, it can be achieved that, ICU proxy can decrease the page load time in all browsers on both Operating systems, but Squid proxy can only decrease the page load time in Firefox on OS X, and in IE on Win7. At this point, this can be concluded that, Squid proxy will not guarantee proxy caching benefits on all operating systems, and in all browsers, while IUC proxy showed how it can achieve benefits on both OS X, and Win7 and in all tested browsers.

4.4 Non-Optimal Case Test and Analysis on 2G, 3G, 4G

In previous sections, the tests results might present the optimal benefits that IUC proxy can bring to a webpage in terms of HTTP request minimization, and Increasing HTTP compression ratio. The designed web shop page; included images that could easily fit to the bundling proxy, so the number of reduced http requests and amount of compressed aggregated data in JS bundle was remarkable. However, when it comes to the real world wide web situations, there could be some factors that can reduce the optimal benefits of IUC proxy such as:

- Some pages might include functions and scripts which will take some specific amount of time to be stopped, regardless of how fast the images are fetched and rendered before the stoppage time of those functions.
- Some pages might include so many large images which do not fit the bundling policy of IUC proxy.
- Some pages might include images with various colors so that they get various Base64 values, and this may decrease the aggregative compression ratio, and also costs more time for compression and decompression.
- IUC proxy method requires some amount of time at proxy to Encode, and Compress images; and some amount of time at the client side to Decompress, and Decode them. When the total size of the JS Bundle increases, more processing time might be required to be spent on Encoding, Compressing, Decompressing and Decoding. So in very fast connections, where the data can be transferred quickly, the more processing time spent on proxy and client side, the less benefits can be achieved from the IUC proxy. In some cases, where the connection is very fast and the size of images that can be bundled is big, the IUC proxy might also increase the page load time instead of decreasing it.
- Some pages also have the “Lazy Loading” feature, which only requests and shows the images to the client when s/he scrolled down the browser window to view them. In such cases there might be not much image for IUC proxy to bundle and compress, and in fast connections might also stop IUC proxy to show its benefits because the processing time might cost equally or even more than the gains from removing some small amount of HTTP requests.

In this section, a real non-optimal world wide web test is conducted in order to see the practical benefits from the user experience perspective with a non-optimal case. For this reason, from Finland top 100 Alexa site, “yle.fi” with the rank of 8, was selected. The site has no page size variation for a single web page; which provides stability in data transfer size and reliability in tests results. Also “yle.fi” has the previously mentioned “Lazy Loading” feature which gives the page this ability not to request and load those images that are located in non-scrolled down part of the page; which could be a very challenging issue and non-optimal case for the IUC proxy as the proxy can unify and compress less images in a single JS bundle.

In the following, in order to also examine the IUC proxy benefits for Mobile QoS, the results are reported and analyzed based on the tests conducted through 2G, 3G, and 4G Networks bandwidth provided by an iPhone 5s Hotspot via Wi-Fi to a MacBook Pro with OS X EI Capitan in Chrome “51.0.2704.103”. Chrome browser is selected in this test because both IUC proxy and Squid proxy has shown a constant behavior and trend on this browser both on Win7 and OS X via different connection types. The reported loading time and data transfer results are the average amounts of gathered values from 25 repetitions per each test Setup.

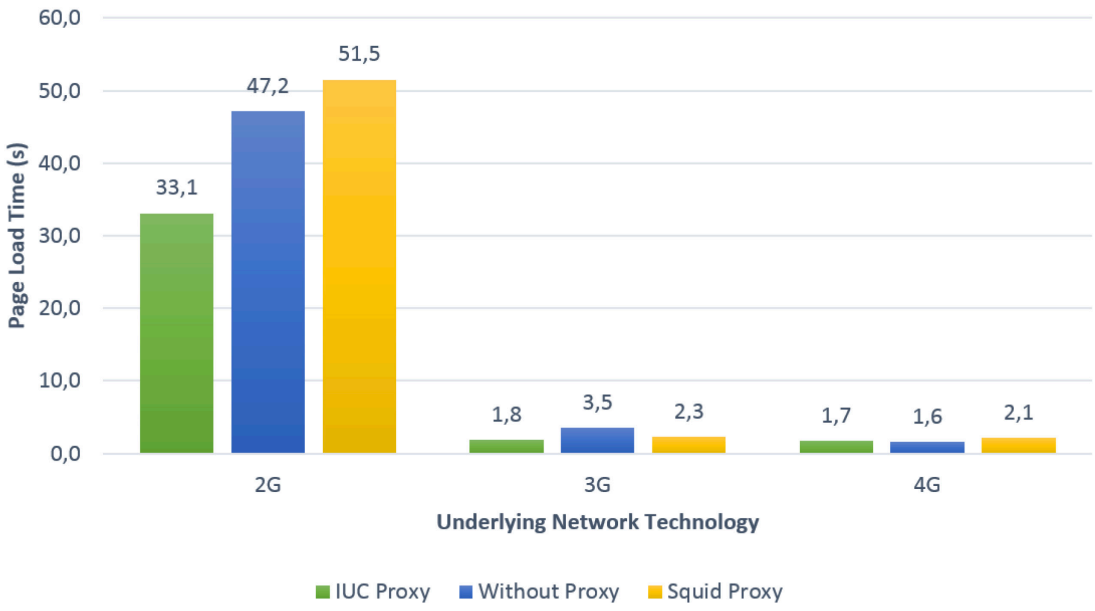

Figure 21: “Yle.fi” Loading Time results via 2G, 3G, and 4G on OSX in Chrome

Figure 21 illustrates that IUC proxy can decrease the page load time by 35 % with a 2G bandwidth which means saving around 18 seconds. On the other side, the Squid proxy not only didn’t decrease the loading time, but also increased it by 8 % which means adding around 4 seconds of extra loading time.

When it comes to 3G bandwidth, IUC proxy still decreases the page load time, by 49 % which means saving around 1.7 seconds. On the other side, Squid proxy has also decreased the page loading time by 35 % which means saving around 1.2 seconds which is less than IUC proxy savings.

But, when it comes to a very fast connection like 4G, the IUC proxy stops to show any benefits. The reason behind this issue is explained before in this section, and is all about the fact that in such cases there might be not much image for IUC proxy to bundle and compress, and in fast connections this issue might stop IUC proxy to show its benefits because the processing time costs equally or even more than the gains from removing some small amount of HTTP requests.

On the other side of the game, the Squid proxy not only didn't stop bringing benefits, but also harms the loading time by 31 % which is means adding around 500 milliseconds.

4.5 Data Transfer Efficiency Analysis

In This section the results of the tests on data transfer efficiency, are reported and analyzed. The reported data transfer results are the average amounts of gathered values from 25 repetitions per each test Setup.

The first chart in this section shows the test results achieved from the tests on the designed web shop in order to know the optimum possible compression rates. Figure 22 illustrates that IUC proxy can decrease the data transfer size up to 75 % for the heavy page, surprisingly Squid proxy adds some Kilobytes of data to the whole transfer size on all different page weights.

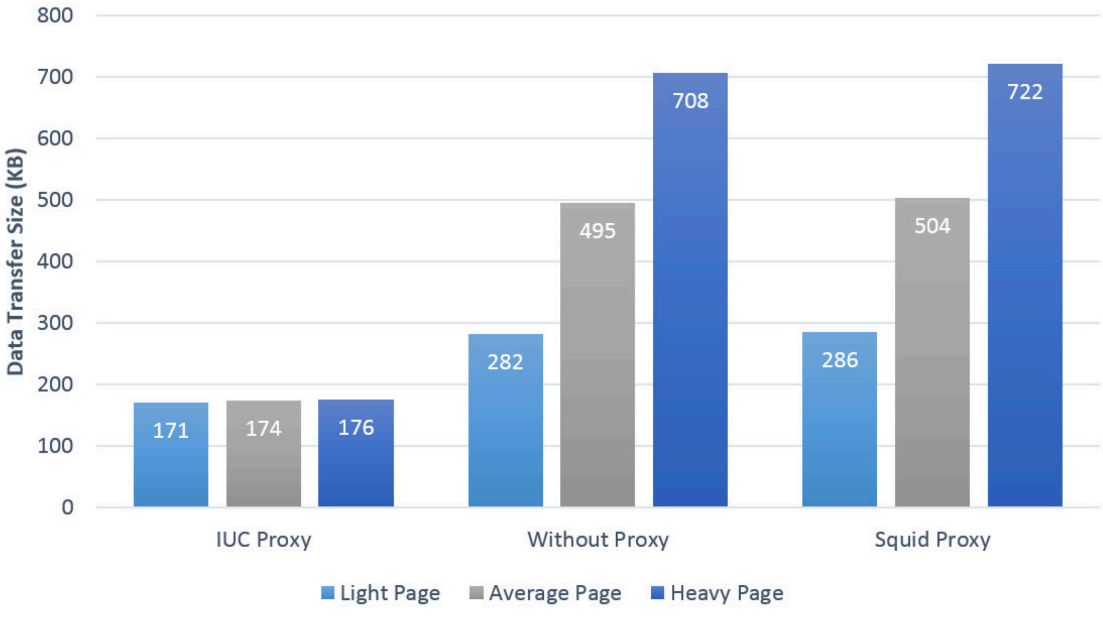

Figure 22: Data Transfer Size Comparison on "Test Webshop"

As we discussed earlier there might be some reasons that IUC proxy stops bringing benefits either to page loading time or the data transfer size. One of those reasons was those pages which have small number of images, or web pages which have the “Lazy Loading” feature in order to show only a small number of images which are located at the most top viewable part of the page out of all the page images. In such case the compression ratio will not be so remarkable or also may be completely vanished as in some cases the saved amount of data might be equal or less than the Base64 encoding overhead.

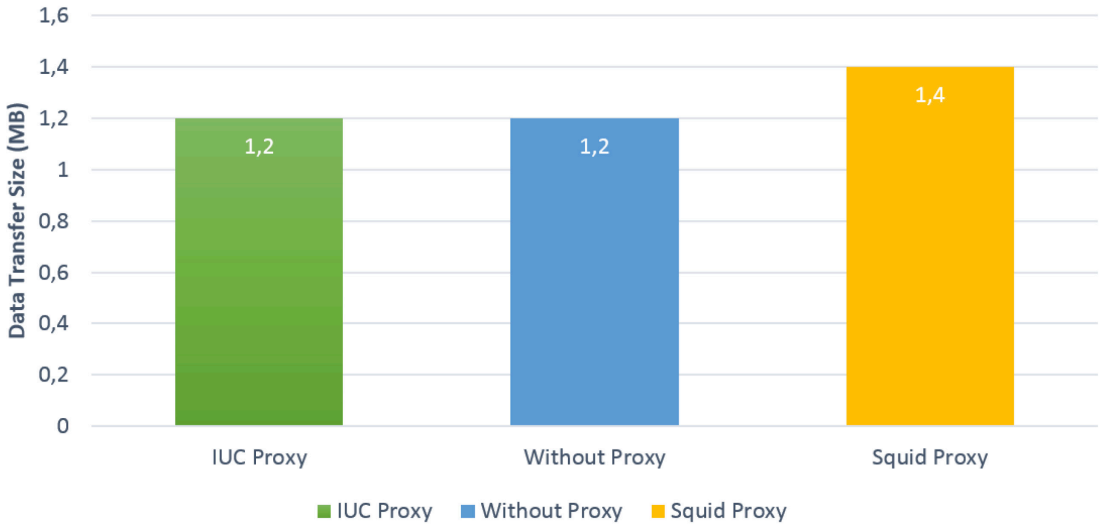

Figure 23: Data Transfer Size Comparison on “yle.fi”

Figure 23 shows how “yle.fi” home page which uses the “Lazy Loading” feature, stopped to make benefit out of IUC aggregative HTTP compression method. This can make us to conclude that the only factor that brought benefits to “yle.fi” home page faster loading through IUC proxy with 2G, and 3G bandwidth capacities; might be reducing the number of HTTP requests.

On the other side of the game, game the surprising point is how Squid proxy adds 0.2 Megabytes of data to the overall transfer size; which might be one of the reasons for adding extra time to the page loading time.

4.6 SEO Affection Analysis and Considerations

When IUC proxy is deployed as a “B2B” service, it can bring benefit to the web owners as a CDN in terms of transfer size and transfer time, and at the same time by improving their “webpages” performance at the client side; which will give them a remarkable SEO advantage on the web among other webpages which are not equipped with IUC technology. But the question might be beside the speed, “Is there any other aspect that IUC proxy can affect either negatively or positively in terms of SEO?”. In the following

some probable cases are described briefly to make an overall picture of other possible SEO affections by IUC proxy.

- Increasing page size hurts SEO and leads to lower crawl rates. Dealing with this fact, it should be considered that IUC proxy stores all image data in a separate JS file; which makes the images data separate from the HTML page, and subsequently does not increases the page size. In addition to that, IUC JS Cache Functions and JS linked files are great solutions to the problem of code bloat. Code bloat occurs when the size of an HTML file used for a web page approaches the limits set by the search engines.
- Serving more bytes in overall might hurt SEO. Dealing with this fact, it should be noted that, IUC proxy benefits from its aggregative HTTP compression which reduces the data transfer size significantly in each go, and at the same time it benefits from the ISCS-Cache (IUC Stateful Caching System); which is designed to keep track of the stored and cached textual data of images at client side to maintain benefits from client side caching.

- Base64 images are not indexed by search engines and won't appear in image search results. Dealing with this fact, it should be considered that Images should have enough high quality to rank in image search, and "high quality" means "big size". IUC proxy has encoding policies which enables the content provider to encode images with a specific size limit. By default, this limit is set to 40KB (for small size images like small size thumbnails of catalogues); which means that IUC does not encode "big size" images which might have a chance to be ranked in image search. Even if there is a small size image in a page that ranks well in the image search; then IUC can be informed about that and be configured not to encode that specific low size and highly ranked image. The following text is extracted from answer of "John Mu" at Google to a question about using Data URIs in HTML Pages: "If these are tiny thumbnails that nobody searches for, then that might be a possibility. If you want to keep the larger images, you could also just keep those images, and only embed the thumbnails as data URIs." [39] As a practical example to this fact, the search results for the term "shop dresses in Finland" in Google for "Marimekko" related images as potential web owner, only resulted in 2 related images in top 100 results with the sizes of 312KB, and 106KB. As a result, none of them will fit to the IUC default encoding policy, and therefore those indexed images can be crawled and indexed successfully by search engines.

- Search engines like Google might not be able to execute the IUC JavaScript codes and the inserted data from the linked JS files. Dealing with this assumption it should be told that Google is able to execute and index all kind of JavaScript codes, and it is also able to render the entire page and read the DOM. Any content that is dynamically inserted in the DOM, is also crawlable and indexable by search engines like Google. [40] All the following elements such as "JavaScript Redirects", "JS Links", "Dynamically Inserted Content", and "Dynamically Inserted Metadata" can be easily crawled and indexed by Google.

- Many JavaScript linked files to go through and follow, might slow down the crawling. Dealing with this fact, it should be noticed that IUC proxy adds rel=”nofollow” to <script> tags of the JavaScript linked files. This way the search engine spider will not crawl to that JS file, and can fastly crawl through more valuable content of the HTML page.

- Using rel=“nofollow” attribute in JS linked files <script> tags, might hurt SEO. Dealing with this assumption; it should be considered that with regard to Google Webmaster Guidelines; rel=”nofollow” attribute is forbidden for <a> tags and no other tags like <script>. [41]
- Adding more than 100 JS linked files to an HTML page might hurt SEO in search engines like Google. Dealing with such assumption; it should be noted that Google once had a guideline where its search engine spider (Googlebot) would not crawl and index a page if it detected more than 100 links. But now, Google and other search engines has removed the 100 links per page limit, since the webpages are getting bigger with more rich media.

Other cases or problems with the proposed method might be available but for now they are not clear to the author of this. As a result, there is no 100 % guarantee of unavailability of any negative impact from SEO perspective on B2B implementation of the proposed IUC proxy method.

5 Summary

As mentioned before in this thesis, web content delivery optimization could be one of the oldest web optimization issues with various old, current and coming solutions. It brings benefits both to end users and web owners. End users always seek for a better User Experience and on the other hand web owners always seek for higher Hit Rate and Conversion Rate in order to earn more, and at the same time save more if possible. Major categories of server side optimization methods in web content delivery might be recognized as; “HTTP requests Minimization”, “HTTP Compression”, and “Server Side Caching”. The proposed method in this thesis tries to propose new methods within the two categories of “HTTP requests Minimization”, and “HTTP Compression”; mainly for slow and fare connections.

The method utilizes a proxy; called IUC (Image Unifier and Compressor) as a middle box to fetch the content requested by a client, from a single or multiple web servers, and bundles all of the fetched image content types that fits the bundling policy; inside a JavaScript file in base64 format. The proxy equips the HTML page; linking to the bundled images, with proxy assigned “IDs” and “Remote Cache Function” for images, to handle the tracking of the bundled images and ensuring the maintenance of the client side caching benefits. If any compression scheme is supported by client; then HTTP compression is also applied to all the content before delivering them to the client.

This optimization method reduces the number of HTTP requests between the client and multiple web servers as a result of its proposed bundling method for image content type, and at the same time it improves the HTTP compression efficiency as a result of its proposed method of aggregative textual content compression. The proxy can be used both as a “reverse proxy” (B2B model), and as a “forward proxy” (B2C model).

Conducted tests results in real world wide web environment with a non-optimal webpage, showed that IUC as a forward proxy can effectively reduce the page load time from 35 % to 50 % in slow and decent network bandwidths like 2G, and 3G (0.2 Mbps - 15 Mbps downlink). However, the proposed method cannot guarantee any improvement with non-optimal cases on fast connections such as 4G, Gigabit Ethernet, or fast Wi-Fi; for non-optimal cases because of various web content cases mentioned in section 4.4 of this thesis, that might reduce or vanish the benefits of the proposed proxy method.

Conducted test results on the developed test web shop; which was intended for creating reliable testing condition and finding the optimal measurement results, has proved up to 81 % faster page loading time, and also 75 % less data transfer size. This results might also be interesting from a reverse proxy perspective where, web owners can adapt their content and web design structure to make the maximum benefit out of the IUC proxy method, once they are one hundred percent sure about the SEO considerations of IUC method. The following tables can briefly explain the summary of the results.

Optimization Index	Safari 840 Mbps dl	Chrome 840 Mbps dl	Firefox 840 Mbps dl	IE 840 Mbps dl
Maximum Page Loading Time Acceleration Ratio	73 %	81 %	60 %	64 %
Maximum Data Transfer Saving Ratio	75 %			

Table 3: IUC benefits with optimal webpage adaptability

Optimization Index	2G 0.2 Mbps	3G 15 Mbps	4G 32 Mbps	Fast Wi-Fi 150 Mbps	Gigabit Ethernet 850 Mbps
Maximum Recorded Page Loading Time Acceleration Ratio	35 %	50 %	Not Achieved	Not Achieved	Not Achieved
Maximum Recorded Data Transfer Saving Ratio	25 %				

Table 4: IUC benefits with non-optimal webpage adaptability

Although the proposed method in this thesis has shown a beneficial trend in terms of decreasing the data transfer size and page loading time, this fact should also be taken into account that, the page loading time can be affected by multiple factors which can make the whole page loading procedure to ignore how fast or how much compressed the image content is delivered to the client, or even to stop making benefit out of bundling and compression. Continuous study of the behavior and development structure of popular websites can make the available solutions in the proposed method more mature in order to bring benefit to a wider range of pages on the web.

References

[1] "Managing Network ResourcesUser," in Guide for the Cisco Secure Access Control System 5.1. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/net_mgmt/cisco_secure_access_control_system/5-1/user/guide/acsuserguide/net_resources.pdf. Accessed: Jul. 31, 2016.

[2] S. Wang, T. Lei, L. Zhang, C.-H. Hsu, and F. Yang, "Offloading mobile data traffic for QoS-aware service provision in vehicular cyber-physical systems," *Future Generation Computer Systems*, vol. 61, pp. 118–127, Aug. 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X15003179. Accessed: Aug. 3, 2016.

[3] C. Quadros, A. Santos, M. Gerla, and E. Cerqueira, "QoE-driven dissemination of real-time videos over vehicular networks," *Computer Communications*, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366416302651. Accessed: Aug. 3, 2016.

[4] G. Egri and C. Bayrak, "The role of search engine optimization on keeping the user on the site," *Procedia Computer Science*, vol. 36, pp. 335–342, Jan. 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050914013519. Accessed: Aug. 3, 2016.

[5] K. Marlow, "The importance of Website loading speed & top 3 factors that limit Website speed," in hosting, Aykira Internet Services, 2014. [Online]. Available: http://www.aykira.com.au/2014/04/importance-website-loading-speed-top-3-factors-limit-website-speed/. Accessed: Aug. 1, 2016.

[6] T. Everts, "Case study: How a 2-Second improvement in page load time more than doubled conversions," in radware, Radware Blog, 2013. [Online]. Available: http://blog.radware.com/applicationdelivery/applicationacceleration optimization/2013/05/case-study-page-load-time-conversions/. Accessed: Aug. 1, 2016.

[7] "Slow-loading websites cost retailers £1.73bn in lost sales each year," Econsultancy, 2012. [Online]. Available: https://econsultancy.com/blog/9790-slow-loading-websites-cost-retailers-1-73bn-in-lost-sales-each-year. Accessed: Aug. 1, 2016.

[8] "New study quantifies the impact of broadband speed on GDP," in ericsson, Ericsson.com, 1970. [Online]. Available: http://www.ericsson.com/news/1550083. Accessed: Aug. 1, 2016.

[9] Cambridge Dictionary, "Quality meaning in the Cambridge English dictionary," in cambridge, © Cambridge University Press, 2015. [Online]. Available: http://dictionary.cambridge.org/dictionary/english/quality. Accessed: Aug. 1, 2016.

[10] M. Ashjaei, M. Behnam, L. Almeida, and T. Nolte, "MTU configuration for real-time switched Ethernet networks," *Journal of Systems Architecture*, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1383762116300261. Accessed: Aug. 3, 2016.

[11] Divestopedia and S. Institute, "What is quality of service (QoS)? - definition from Techopedia," in technopedia, Techopedia.com, 2016. [Online]. Available: https://www.techopedia.com/definition/9049/quality-of-service. Accessed: Aug. 1, 2016.

[12] J.-J. Huang, P.-C. Shen, and Y.-T. Tseng, "Enhancing performance of feedback-based QoS scheduling for video delivery over WLANs," *Computer Communications*, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366416302602. Accessed: Aug. 3, 2016.

[13] G. Miao; J. Zander; K-W Sung; B. Slimane (2016). Fundamentals of Mobile Data Networks. Cambridge University Press. ISBN 1107143217.

[14] A. Dudin, C. Kim, S. Dudin, and O. Dudina, "Analysis and optimization of guard channel policy with buffering in cellular mobile networks," *Computer Networks*, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128616301013. Accessed: Aug. 3, 2016.

[15] Hermes Irineu Del Monego1, Eliane Lucia Bodanese2, Luiz Nacamura Jr1, and Richard Demo Souza1, "A Dynamic Resource Allocation Scheme for Providing QoS in Packet-Switched Cellular Networks," http://springerlink.metapress.com/(xxolql45nijqxu55b4eq5345)/app/home/main.asp?referrer=default

[16] S. Tanwir and H. Perros, "Modeling live adaptive streaming over HTTP," *Computer Communications*, vol. 85, pp. 74–88, Jul. 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366416301153. Accessed: Aug. 3, 2016.

[17] K. Kalajdzic, S. H. Ali, and A. Patel, "Rapid lossless compression of short text messages," *Computer Standards & Interfaces*, vol. 37, pp. 53–59, Jan. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0920548914000737. Accessed: Aug. 3, 2016.

[18] "Optimizing encoding and transfer size of text-based assets | web fundamentals - Google developers," in Google Developers, Optimizing encoding and transfer size of text-based assets | Web Fundamentals - Google Developers, 2014. [Online]. Available:

https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/optimize-encoding-and-transfer?hl=en. Accessed: Aug. 1, 2016.

[19] R. Verborgh *et al.*, "Triple pattern fragments: A low-cost knowledge graph interface for the web," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. s 37–38, pp. 184–206, Mar. 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570826816000214. Accessed: Aug. 3, 2016.

[20] 2013, "An explanation of the 'Deflate' algorithm," 1996. [Online]. Available: http://www.zlib.net/feldspar.html. Accessed: Aug. 1, 2016.

[21] [LZ77] Ziv J., Lempel A., ``A Universal Algorithm for Sequential Data Compression," IEEE Transactions on Information Theory, Vol. 23, No. 3, pp. 337-343.

[22] "RFC 1951 DEFLATE compressed data format specification ver 1.3," 1996. [Online]. Available: https://www.w3.org/Graphics/PNG/RFC-1951. Accessed: Aug. 1, 2016.

[23] T. Mori *et al.*, "Statistical estimation of the names of HTTPS servers with domain name graphs," *Computer Communications*, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366416300068. Accessed: Aug. 3, 2016.

[24] G. Podjarny, "SPDY & HTTP 2," in akamai. [Online]. Available: https://blogs.akamai.com/2014/01/spdy-http-2.html. Accessed: Aug. 1, 2016.

[25] "HTTP/2 frequently asked questions," in Github. [Online]. Available: https://http2.github.io/faq/. Accessed: Aug. 1, 2016.

[26] X. Zhang, N. Wang, V. G. Vassilakis, and M. P. Howarth, "A distributed in-network caching scheme for P2P-like content chunk delivery," *Computer Networks*, vol. 91, pp. 577–592, Nov. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128615002959. Accessed: Aug. 3, 2016.

[27] W. Ali, S. M. Shamsuddin, and A. S. Ismail, "Intelligent web proxy caching approaches based on machine learning techniques," *Decision Support Systems*, vol. 53, no. 3, pp. 565–579, Jun. 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167923612001078. Accessed: Aug. 3, 2016.

[28] E. Baccaglini, M. Grangetto, E. Quacchio, and S. Zezza, "A study of an hybrid CDN–P2P system over the PlanetLab network," *Signal Processing: Image Communication*, vol. 27, no. 5, pp. 430–437, May 2012. [Online]. Available:

http://www.sciencedirect.com/science/article/pii/S0923596512000355. Accessed: Aug. 3, 2016.

[29] Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. "The Akamai Network: A Platform for High-Performance Internet Applications, ACM SIGOPS Operating Systems Review, vol. 44, no. 3, July 2010."

[30] M. Claeys, D. Tuncer, J. Famaey, M. Charalambides, S. Latre, F. De Turck, G. Pavlou, “Proactive Multi-tenant Cache Management for Virtualized ISP Networks,” proceedings of IEEE/IFIP Conference on Network and Service Management (CNSM), Rio de Janeiro, Brazil, November 2014.

[31] A. Technologies, "Web & mobile performance," 2016. [Online]. Available: https://www.akamai.com/us/en/web-and-mobile-performance.jsp. Accessed: Aug. 3, 2016.

[32] CloudFlare, "The web performance & security company,". [Online]. Available: https://www.cloudflare.com/overview/. Accessed: Aug. 1, 2016.

[33] "Data saver - Google chrome," Google+Add us on, 2015. [Online]. Available: https://developer.chrome.com/multidevice/data-compression. Accessed: Aug. 3, 2016.

[34] N. Pahwa, "There’s A business angle to the Hungama-Opera speed dial deal," in *Advertising*, MediaNama, 2011. [Online]. Available: http://www.medianama.com/2011/05/223-theres-a-business-angle-to-the-hungama-opera-speed-dial-deal/. Accessed: Aug. 3, 2016.

[35] "Mini gets mighty: Introducting opera Mini 2.0 for your mobile phone | opera,". [Online]. Available: http://www.operasoftware.com/press/releases/mobile/mini-gets-mighty-introducting-opera-mini-2.0-for-your-mobile-phone. Accessed: Aug. 3, 2016.

[36] L. Wang and J. Manner, "Energy-efficient mobile web in a bundle," Computer Networks, vol. 57, no. 17, pp. 3581–3600, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128613002624?np=y. Accessed: Aug. 1, 2016.

[37] R. Netravali, "Polaris: Faster Page Loads Using Fine-grained Dependency Tracking," in *web.mit.edu*. [Online]. Available: http://web.mit.edu/ravinet/www/polaris_nsdi16.pdf. Accessed: Aug. 3, 2016.

[38] "Base64 decode and Encode - online," in base64, www.BASE64DECODE.org. [Online]. Available: https://www.base64decode.org. Accessed: Aug. 1, 2016.

[39] "Google groups," in Google Product Forum. [Online]. Available: https://productforums.google.com/forum/#!topic/webmasters/MDujtIpTuPo. Accessed: Aug. 1, 2016.

[40] A. Audette, "We tested how Googlebot crawls Javascript and here's what we learned," in All Things SEO Column, Search Engine Land, 2015. [Online]. Available: http://searchengineland.com/tested-googlebot-crawls-javascript-heres-learned-220157. Accessed: Aug. 1, 2016.

[41] D. Schubert, "Influence of mobile-friendly design to search results on Google search," *Procedia - Social and Behavioral Sciences*, vol. 220, pp. 424–433, May 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877042816306188. Accessed: Aug. 3, 2016.